AD NUMBER

**ADB120253**

LIMITATION CHANGES

TO:

**Approved for public release; distribution is unlimited.**

FROM:

**Distribution authorized to U.S. Gov't. agencies only; Critical Technology; MAR 1988. Other requests shall be referred to Air Force Armament Lab., Eglin AFB, FL 32542. This document contains export-controlled technical data.**

AUTHORITY

**AFSC wright lab at eglin afb, FL ltr dtd 13 Feb 1992**

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>Unclassified | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Distribution authorized to U.S. Government Agencies and their contractors; CT (over) |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | 5. MONITORING ORGANIZATION REPORT NUMBER(S)<br>AFATL-TR-88-18, Vol 6 |
| 6a. NAME OF PERFORMING ORGANIZATION<br>McDonnell Douglas Astronautics Company<br>6b. OFFICE SYMBOL (If applicable) | 7a. NAME OF MONITORING ORGANIZATION<br>Aeromechanics Division |
| 6c. ADDRESS (City, State, and ZIP Code)<br>P.O. Box 516<br>St. Louis, MO 63166 | 7b. ADDRESS (City, State, and ZIP Code)<br>Air Force Armament Laboratory<br>Eglin AFB, FL 32542-5434 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION<br>STARS Joint Program Office<br>8b. OFFICE SYMBOL (If applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER<br>F08635-86-C-0025 |
| 8c. ADDRESS (City, State, and ZIP Code)<br>Room 3D139 (1211 Fern St)<br>The Pentagon<br>Washington DC 20301-3081 | 10. SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO.: 63756D; PROJECT NO.: 921C; TASK NO.: GZ; WORK UNIT ACCESSION NO.: 57 |

TITLE (Include Security Classification)
Common Ada Missile Package (CAMP) Project: Missile Software Parts, Vol 6: Top-Level Design Document (Vol 4-6)

PERSONAL AUTHOR(S)
D. McNicholl, S. Cohen, C. Palmer, et al.

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15. PAGE COUNT |
|---|---|---|---|
| Technical Note | FROM Sep 85 TO Mar 88 | March 1988 | 245 |

SUPPLEMENTARY NOTATION
SUBJECT TO EXPORT CONTROL LAWS.
Availability of this report is specified on verso of front cover. (over)

| COSATI CODES | | |
|---|---|---|
| FIELD | GROUP | SUB-GROUP |
| | | |
| | | |

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Reusable Software, Missile Software, Software Generators
Ada, Parts Composition, Systems, Software Parts

ABSTRACT (Continue on reverse if necessary and identify by block number)
The objective of the CAMP program is to demonstrate the feasibility of reusable Ada software parts in a real-time embedded application area; the domain chosen for the demonstration was that of missile flight software systems. This required that the existence of commonality within that domain be verified (in order to justify the development of parts for that domain), and that software parts be designed which address those areas identified. An associated parts system was developed to support parts usage. Volume 1 of this document is the User's Guide to the CAMP Software parts; Volume 2 is the Version Description Document; Volume 3 is the Software Product Specification; Volumes 4-6 contain the Top-Level Design Document, and, Volumes 7-12 contain the Detail Design Documents.



| | |
|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT<br>☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION<br>Unclassified |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>Christine Anderson | 22b. TELEPHONE (Include Area Code)<br>(904) 882-2961<br>22c. OFFICE SYMBOL<br>AFATL/FXG |

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

3. DISTRIBUTION/AVAILABILITY OF REPORT (CONCLUDED)

~~this report documents test and evaluation~~; distribution limitation applied March 1988. Other requests for this document must be referred to AFATL/FXG, Eglin AFB, Florida 32542-5434.

16. SUPPLEMENTARY NOTATION (CONCLUDED)

These technical notes accompany the CAMP final report AFATL-TR-85-93 (3 Vols)

AFATL-TR-88-18, Vol 6

# SOFTWARE TOP LEVEL DESIGN DOCUMENT

FOR THE

MISSILE SOFTWARE PARTS

OF THE

COMMON ADA MISSILE PACKAGE (CAMP) PROJECT

CONTRACT F08635-86-C-0025

CDRL SEQUENCE NO. C014

30 OCTOBER 1987



AIR FORCE ARMAMENT LABORATORY
Air Force Systems Command ▮ United States Air Force ▮ Eglin Air Force Base, Florida

3.6.8.5 DATA CONVERSION



(This page left intentionally blank.)

3.6.8.5.1 UNIT_CONVERSIONS TLCSC (CATALOG #P579-0)

This part, which is a package of generic packages, provides a set of functions which convert data objects from one unit of measurement to another.

3.6.8.5.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R105.

3.6.8.5.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

Each of the generic packages contained in this part requires two generic formal types. These two types are used to define the units on which the conversions are to take place.

3.6.8.5.1.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.8.5.1.4 LOCAL ENTITIES

None.

3.6.8.5.1.5 INTERRUPTS

None.

3.6.8.5.1.6 TIMING AND SEQUENCING

The following is a sample usage of one of the LLCSC's contained in this part. The other parts would be used in a similar manner.

```
with Basic_Data_Types; use Basic_Data_Types;
with Unit_Conversions;
...
   package BDT   renames Basic_Data_Types;
   package UConv renames Unit_Conversions;
   ...
   package MF_Convert is new
             UConv.Meters_and_Feet (Meters => BDT.Meters,
                                    Feet   => BDT.Feet);
   ...
   X_Feet  : BDT.Feet;
   X_Meter : BDT.Meters;
   ...
      begin
         ...
```

```
X_Feet := MF_Convert.Conversion_to_Feet(Input => X_Meters);
...
```

3.6.8.5.1.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.8.5.1.8 DECOMPOSITION

The following table describes the decomposition of this part, along with units each part deals with. Each package contains one function which goes from Type A units to Type B units, and a second function which goes from Type B units to Type A units. Each of the parts listed is a generic package.

| Name | Type A | Type B |
|---|---|---|
| Meters_and_Feet | Meters | Feet |
| Meters_and_Feet_per_Second | Meters_per_Second | Feet_per_Second |
| Meters_and_Feet_per_Second_Squared | Meters_per_Second_Squared | Feet_per_Second_Squared |
| Gees_and_Meters_per_Second_Squared | Gees | Meters_per_Second_Squared |
| Gees_and_Feet_per_Second_Squared | Gees | Feet_per_Second_Squared |
| Radians_and_Degrees | Radians | Degrees |
| Radians_and_Degrees_per_Second | Radians_per_Second | Degrees_per_Second |
| Radians_and_Semicircles | Radians | Semicircles |
| Radians_and_Semicircles_per_Second | Radians_per_Second | Semicircles_per_Second |
| Degrees_and_Semicircles | Degrees | Semicircles |
| Degrees_and_Semicircles_per_Second | Degrees_per_Second | Semicircles_per_Second |
| Seconds_and_Minutes | Seconds | Minutes |
| Centigrade_and_Fahrenheit | Centigrade | Fahrenheit |
| Centigrade_and_Kelvin | Centigrade | Kelvin |
| Fahrenheit_and_Kelvin | Fahrenheit | Kelvin |
| Kilograms_and_Pounds | Kilograms | Pounds |
| Kilograms_per_Meter_Squared_and_Pounds_per_Foot_Squared | Kilograms_per_Meters_Squared | Pounds_per_Foot_Squared |

The following table summarizes the allocation of catalog numbers to to this part:

| Part | Catalog # |
|---|---|
| Meters_and_Feet | P580-0 |
| Conversion_to_Feet | P581-0 |
| Conversion_to_Meters | P582-0 |
| Meters_and_Feet_per_Second | P583-0 |
| Conversion_to_Feet_per_Second | P584-0 |
| Conversion_to_Meters_per_Second | P585-0 |
| Meters_and_Feet_per_Second_Squared | P586-0 |
| Conversion_to_Feet_per_Second2 | P587-0 |
| Conversion_to_Meters_per_Second2 | P588-0 |
| Gees_and_Meters_per_Second_Squared | P589-0 |
| Conversion_to_Meters_per_Second2 | P590-0 |
| Conversion_to_Gees | P591-0 |
| Gees_and_Feet_per_Second_Squared | P592-0 |
| Conversion_to_Feet_per_Second2 | P593-0 |
| Conversion_to_Gees | P594-0 |
| Radians_and_Degrees | P595-0 |
| Conversion_to_Degrees | P596-0 |
| Conversion_to_Radians | P597-0 |
| Radians_and_Degrees_per_Second | P598-0 |
| Conversion_to_Degrees_per_Second | P599-0 |
| Conversion_to_Radians_per_Second | P600-0 |
| Radians_and_Semicircles | P601-0 |
| Conversion_to_Semicircles | P602-0 |
| Conversion_to_Radians | P603-0 |
| Radians_and_Semicircles_per_Second | P604-0 |
| Conversion_to_Semicircles_per_Second | P605-0 |
| Conversion_to_Radians_per_Second | P606-0 |
| Degrees_and_Semicircles | P607-0 |
| Conversion_to_Semicircles | P608-0 |
| Conversion_to_Degrees | P609-0 |
| Degrees_and_Semicircles_per_Second | P610-0 |
| Conversion_to_Semicircles_per_Second | P611-0 |
| Conversion_to_Degrees_per_Second | P612-0 |
| Seconds_and_Minutes | P613-0 |
| Conversion_to_Minutes | P614-0 |
| Conversion_to_Seconds | P615-0 |
| Centigrade_and_Fahrenheit | P616-0 |
| Conversion_to_Fahrenheit | P617-0 |
| Conversion_to_Centigrade | P618-0 |
| Centigrade_and_Kelvin | P619-0 |
| Conversion_to_Kelvin | P620-0 |
| Conversion_to_Centigrade | P621-0 |
| Fahrenheit_and_Kelvin | P622-0 |
| Conversion_to_Kelvin | P623-0 |
| Conversion_to_Fahrenheit | P624-0 |
| Kilograms_and_Pounds | P625-0 |
| Conversion_to_Pounds | P626-0 |
| Conversion_to_Kilograms | P627-0 |
| Kilograms_per_Meter_Squared_and_ Pounds_per_Foot_Squared | P628-0 |
| Conversion_to_Pounds_per_Foot2 | P629-0 |
| Conversion_to_Kilograms_per_Meter2 | P630-0 |

3.6.8.5.1.9 PART DESIGN

None.

```
package Unit_Conversions is

-- ------------------------------------
-- --packages involving Meters and Feet-
-- ------------------------------------

-- ----- Meters <==> Feet -----

   generic
      type Meters is digits <>;
      type Feet   is digits <>;
   package Meters_And_Feet is

      function Conversion_To_Feet (Input : Meters) return Feet;

      function Conversion_To_Meters (Input : Feet) return Meters;

   end Meters_And_Feet;

-- ----- Feet/Second <==> Meters/Second -----

   generic
      type Feet_Per_Second   is digits <>;
      type Meters_Per_Second is digits <>;
   package Meters_And_Feet_Per_Second is

      function Conversion_To_Feet_Per_Second
                  (Input : Meters_Per_Second) return Feet_Per_Second;

      function Conversion_To_Meters_Per_Second
                  (Input : Feet_Per_Second) return Meters_Per_Second;

   end Meters_And_Feet_Per_Second;

-- ----- Feet/Second**2 <==> Meters/Second**2 -----

   generic
      type Feet_Per_Second_Squared   is digits <>;
      type Meters_Per_Second_Squared is digits <>;
   package Meters_And_Feet_Per_Second_Squared is

      function Conversion_To_Feet_Per_Second2
                  (Input : Meters_Per_Second_Squared) return Feet_Per_Second_Squared;

      function Conversion_To_Meters_Per_Second2
                  (Input : Feet_Per_Second_Squared) return Meters_Per_Second_Squared;

   end Meters_And_Feet_Per_Second_Squared;

-- --------------------------
-- --packages involving Gees-
-- --------------------------

-- ----- Gees <==> Meters/Second**2 -----

   generic
      type Gees                        is digits <>;
```

```
      type Meters_Per_Second_Squared is digits <>;
   package Gees_And_Meters_Per_Second_Squared is

      function Conversion_To_Meters_Per_Second2
                  (Input : Gees) return Meters_Per_Second_Squared;

      function Conversion_To_Gees
                  (Input : Meters_Per_Second_Squared) return Gees;

   end Gees_And_Meters_Per_Second_Squared;

-- ----- Gees <==> Feet/Second**2 -----

   generic
      type Gees                     is digits <>;
      type Feet_Per_Second_Squared is digits <>;
   package Gees_And_Feet_Per_Second_Squared is

      function Conversion_To_Feet_Per_Second2
                  (Input : Gees) return Feet_Per_Second_Squared;

      function Conversion_To_Gees
                  (Input : Feet_Per_Second_Squared) retu

   end Gees_And_Feet_Per_Second_Squared;

-- ----------------------------------------
-- --packages involving Radians and Degrees-
-- ----------------------------------------

-- ----- Radians <==> Degrees -----

   generic
      type Radians is digits <>;
      type Degrees is digits <>;
   package Radians_And_Degrees is

      function Conversion_To_Degrees (Input : Radians) return Degrees;

      function Conversion_To_Radians (Input : Degrees) return Radians;

   end Radians_And_Degrees;

-- ----- Radians/Second <==> Degrees/Second -----

   generic
      type Radians_Per_Second is digits <>;
      type Degrees_Per_Second is digits <>;
   package Radians_And_Degrees_Per_Second is

      function Conversion_To_Degrees_Per_Second
                  (Input : Radians_Per_Second) return Degrees_Per_Second;

      function Conversion_To_Radians_Per_Second
                  (Input : Degrees_Per_Second) return Radians_Per_Second;

   end Radians_And_Degrees_Per_Second;
```

```
-- ------------------------------------------
-- --packages involving Radians and Semicircles-
-- ------------------------------------------

-- ----- Radians <===> Semicircles -----

   generic
      type Radians     is digits <>;
      type Semicircles is digits <>;
   package Radians_And_Semicircles is

      function Conversion_To_Semicircles (Input : Radians) return Semicircles;

      function Conversion_To_Radians (Input : Semicircles) return Radians;

   end Radians_And_Semicircles;

-- ----- Radians/Second <===> Semicircles/Second -----

   generic
      type Radians_Per_Second     is digits <>;
      type Semicircles_Per_Second is digits <>;
   package Radians_And_Semicircles_Per_Second is

      function Conversion_To_Semicircles_Per_Second
                  (Input : Radians_Per_Second) return Semicircles_Per_Second;

      function Conversion_To_Radians_Per_Second
                  (Input : Semicircles_Per_Second) return Radians_Per_Second;

   end Radians_And_Semicircles_Per_Second;

-- ------------------------------------------
-- --packages involving Degrees and Semicircles-
-- ------------------------------------------

-- ----- Degrees <===> Semicircles -----

   generic
      type Degrees     is digits <>;
      type Semicircles is digits <>;
   package Degrees_And_Semicircles is

      function Conversion_To_Semicircles (Input : Degrees) return Semicircles;

      function Conversion_To_Degrees (Input : Semicircles) return Degrees;

   end Degrees_And_Semicircles;

-- ----- Degrees/Second <===> Semicircles/Second -----

   generic
      type Degrees_Per_Second     is digits <>;
      type Semicircles_Per_Second is digits <>;
   package Degrees_And_Semicircles_Per_Second is
```

```
      function Conversion_To_Semicircles_Per_Second
                  (Input : Degrees_Per_Second)
                  return Semicircles_Per_Second;

      function Conversion_To_Degrees_Per_Second
                  (Input : Semicircles_Per_Second)
                 return Degrees_Per_Second;

   end Degrees_And_Semicircles_Per_Second;

-- ----------------------------------------
-- --packages involving Seconds and Minutes-
-- ----------------------------------------

-- ----- Seconds <==> Minutes -----

   generic
      type Seconds is digits <>;
      type Minutes is digits <>;
   package Seconds_And_Minutes is

      function Conversion_To_Minutes (Input : Seconds) return Minutes;

      function Conversion_To_Seconds (Input : Minutes) return Seconds;

   end Seconds_And_Minutes;

-- ------------------------------------------------
-- --packages involving Centigrade and Fahrenheit-
-- ------------------------------------------------

-- ----- Centigrade <==> Fahrenheit -----

   generic
      type Centigrade is digits <>;
      type Fahrenheit is digits <>;
   package Centigrade_And_Fahrenheit is

      function Conversion_To_Fahrenheit
                  (Input : Centigrade) return Fahrenheit;

      function Conversion_To_Centigrade
                  (Input : Fahrenheit) return Centigrade;

   end Centigrade_And_Fahrenheit;

-- ----------------------------------------
-- --packages involving Centigrade and Kelvin-
-- ----------------------------------------

-- ----- Centigrade <==> Kelvin -----

   generic
      type Centigrade is digits <>;
      type Kelvin     is digits <>;
   package Centigrade_And_Kelvin is
```

```
      function Conversion_To_Kelvin (Input : Centigrade) return Kelvin;

      function Conversion_To_Centigrade (Input : Kelvin) return Centigrade;

   end Centigrade_And_Kelvin;

-- ----------------------------------------
-- --packages involving Fahrenheit and Kelvin-
-- ----------------------------------------

-- ----- Fahrenheit <==> Kelvin -----

   generic
      type Fahrenheit is digits <>;
      type Kelvin     is digits <>;
   package Fahrenheit_And_Kelvin is

      function Conversion_To_Kelvin (Input : Fahrenheit) return Kelvin;

      function Conversion_To_Fahrenheit (Input : Kelvin) return Fahrenheit;

   end Fahrenheit_And_Kelvin;

-- ----------------------------------------
-- --packages involving Pounds and Kilograms-
-- ----------------------------------------

-- ----- Kilograms <==> Pounds -----

   generic
      type Kilograms is digits <>;
      type Pounds    is digits <>;
   package Kilograms_And_Pounds is

      function Conversion_To_Pounds (Input : Kilograms) return Pounds;

      function Conversion_To_Kilograms (Input : Pounds) return Kilograms;

   end Kilograms_And_Pounds;

-- ----- Kilograms/Meters**2 <==> Pounds/Foot2 -----

   generic
      type Kilograms_Per_Meter_Squared is digits <>;
      type Pounds_Per_Foot_Squared     is digits <>;
   package Kilograms_Per_Meter_Squared_And_Pounds_Per_Foot_Squared is

      function Conversion_To_Pounds_Per_Foot2
                  (Input : Kilograms_Per_Meter_Squared)
                  return Pounds_Per_Foot_Squared;

      function Conversion_To_Kilograms_Per_Meter2
                  (Input : Pounds_Per_Foot_Squared)
                  return Kilograms_Per_Meter_Squared;

   end Kilograms_Per_Meter_Squared_And_Pounds_Per_Foot_Squared;
```

```
end Unit_Conversions;
```

### 3.6.8.5.2 EXTERNAL_FORM_CONVERSION_TWOS_COMPLEMENT (PACKAGE) TLCSC (CATALOG #P683-0)

This generic package performs scaling operations on input values. It is able to convert two's complement engineering units to floating point representations and to convert floating point to engineering units.

NOTE: The scaled values, while representing two's complement values, are themselves one's complement values and, therefore, are always positive.

The calculations to go from a scaled integer value to an unscaled floating point value are as follows:

```
unscaled_output := unscaled_bias + ((scaled_value - scale_factor_2) *
                                    unscaled_range / scale_factor_1)
```

and the calculations to go from an unscaled floating point value to a scaled integer are as follows:

```
scaled_output := (unscaled_value - unscaled_bias) *
                 (scale_factor_1 / unscaled_range) + scale_factor_2
```

where:

```
scale_factor_1 := 2 ** initial_engineering_units_bits - 1
  (represents the value range which may be assumed by the
   scaled, integer values)
scale_factor_2 := 2 ** (initial_engineering_units_bits - 1)
  (represents the scaled bias; i.e., the amount by which
  the minumum scaled, integer value is negatively offset
  from 0)
unscaled_bias  := [(unscaled_max - unscaled_min + lsb_value) / 2] +
                   unscaled_min
  (represents the offset from 0 of the median unscaled value)
unscaled_range := unscaled_max - unscaled_min
  (represents the value range which may be assumed by the
   unscaled, floating point values)
```

This part raises a NUMERIC_ERROR exception if Initial_Min_Unscaled_Value is greater than Initial_Max_Unscaled_Value.

#### 3.6.8.5.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R106

#### 3.6.8.5.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Scaled_Integers | integer type | Defines scaled variables stored in engineering units |
| Unscaled_Floats | floating point type | Defines unscaled variables stored in floating point format |

Data objects:

The following table summarizes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Bits_In_Unscaled_Values | POSITIVE | Number of significant bits in the engineering units representation |
| Initial_Min_Unscaled_Value | Unscaled_Floats | Minimum value which the unscaled values may assume |
| Initial_Max_Unscaled_Value | Unscaled_Floats | Maximum value which the unscaled values may assume |

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Multiplication operator defining the operation: Scaled_Integers * Unscaled_Floats => Unscaled_Floats |
| "*" | function | Multiplication operator defining the operation: Unscaled_Floats * Unscaled_Floats => Scaled_Integers |
| "/" | function | Division operator defining the operation: Unscaled_Floats / Scaled_Integers => Unscaled_Floats |
| "/" | function | Division operator defining the operation: Unscaled_Floats / Unscaled_Floats => Scaled_Integers |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following table summarizes the data types exported by this part:

| Name | Range | Description |
|---|---|---|
| Positive_Scaled_Integers | 0 .. Max_Scaled_Value | Subtype of generic formal type Scaled_Integers; used to ensure the values of the scaled input parameters are within the allowable range |
| Valid_Unscaled_Floats | Min_Unscaled_Value .. Max_Unscaled_Value | Subtype of generic formal type Unscaled_Floats; used to ensure the values of the scaled input parameters are within the allowable range |

Data objects:

The following table summarizes the data objects exported by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Max_Scaled_Value | Scaled_Integers | 2**Bits_in_Scaled_Values - 1 | Maximum scaled value |
| Value_Range | Unscaled_Floats | Initial_Max_Unscaled_Value - Initial_Min_Unscaled_Value | Range of values which may be assumed by the unscaled values |
| LSB_Value | Unscaled_Floats | Initial_Value_Range / Max_Scaled_Value | Value of the least significant bit in the scaled values |
| Min_Unscaled_Value | Unscaled_Floats | Initial_Min_Unscaled_Value | Minimum unscaled value |
| Max_Unscaled_Value | Unscaled_Floats | Initial_Max_Unscaled_Value | Maximum unscaled value |

3.6.8.5.2.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.8.5.2.4 LOCAL ENTITIES

None.

3.6.8.5.2.5 INTERRUPTS

None.

3.6.8.5.2.6 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with External_Form_Conversion_Twos_Complement;
...
   function "*" (Left  : in POSITIVE;
                 Right : in FLOAT) return FLOAT;
   function "*" (Left  : in FLOAT;
                 Right : in FLOAT) return POSITIVE;
   function "/" (Left  : in FLOAT;
                 Right : in FLOAT) return POSITIVE;
   function "/" (Left  : in FLOAT;
                 Right : in POSITIVE) return FLOAT;
   ...
   package Form_Conversion is new
            External_Form_Conversion_Twos_Complement
               (Scaled_Integers            => POSITIVE,
                Unscaled_Floats            => FLOAT,
                Bits_In_Unscaled_Values    => 8,
                Initial_Min_Unscaled_Value => -250.0,
                Initial_Max_Unscaled_Value => 250.0);
   ...
   Scaled_Integer : Form_Conversion.Positive_Scaled_Integers;
   Unscaled_Float : Form_Conversion.Valid_Unscaled_Floats;
   ...
      begin
         ...
         Unscaled_Float := Form_Conversion.Unscale(Scaled_Integer);
         ...
         Scaled_Float   := Form_Conversion.Scale(Unscaled_Float);
```

3.6.8.5.2.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.8.5.2.8 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Scale | function | Performs scaling operation on a floating point value to convert it to an engineering units representation |
| Unscale | function | Performs an unscaling operation on a value in engineering units representation to convert it to a floating point representation |

The following table lists the catalog part numbers for the decomposition of this part:

| Name | Catalog # |
|---|---|
| Scale | P687-0 |
| Unscale | P688-0 |

3.6.8.5.2.9 PART DESIGN

None.

(This page lef. intentionally blank.)

```
generic
   type Scaled_Integers       is range <>;
   type Unscaled_Floats       is digits <>;
   Bits_In_Scaled_Values      : in POSITIVE;
   Initial_Min_Unscaled_Value : in Unscaled_Floats;
   Initial_Max_Unscaled_Value : in Unscaled_Floats;
   with function "*" (Left  : in Scaled_Integers;
                      Right : in Unscaled_Floats) return Unscaled_Floats is <>;
   with function "*" (Left  : in Unscaled_Floats;
                      Right : in Unscaled_Floats) return Scaled_Integers is <>;
   with function "/" (Left  : in Unscaled_Floats;
                      Right : in Unscaled_Floats) return Scaled_Integers is <>;
   with function "/" (Left  : in Unscaled_Floats;
                      Right : in Scaled_Integers) return Unscaled_Floats is <>;
package External_Form_Conversion_Twos_Complement is

-- --constant definitions

   Max_Scaled_Value : constant Scaled_Integers := 2**Bits_In_Scaled_Values - 1;

   Value_Range      : constant Unscaled_Floats := Initial_Max_Unscaled_Value -
                                                  Initial_Min_Unscaled_Value;

   Lsb_Value        : constant Unscaled_Floats := Value_Range /
                                                  Max_Scaled_Value;

   Min_Unscaled_Value : constant Unscaled_Floats := Initial_Min_Unscaled_Value;
   Max_Unscaled_Value : constant Unscaled_Floats := Initial_Max_Unscaled_Value;

-- --subtype definitions

   subtype Positive_Scaled_Integers is Scaled_Integers
                                  range 0 .. Max_Scaled_Value;

   subtype Valid_Unscaled_Floats is Unscaled_Floats
                             range Min_Unscaled_Value .. Max_Unscaled_Value;

-- --function specifications

   function Scale (Unscaled_Value : Valid_Unscaled_Floats)
                  return Positive_Scaled_Integers;

   function Unscale (Scaled_Value : Positive_Scaled_Integers)
                    return Valid_Unscaled_Floats;

end External_Form_Conversion_Twos_Complement;
```

(This page left intentionally blank.)

### 3.6.8.6 SIGNAL_PROCESSING TLCSC (CATALOG #P70-0)

This package provides signal processing parts. Each part is designed as an Ada generic package, where the generic parameters will specify the data types of the input and output signals and the values for coefficients used in performing the signal processing functions.

#### 3 6.8.6.1 REQUIREMENTS ALLOCATION

The following diagram summarizes the allocation of CAMP requirements to this part's LLCSC's.

| Name | Type | Req. Allocation |
|---|---|---|
| Limiter (Upper & Lower Bounds) | generic package | R108 |
| Limiter (Upper Bound) | generic package | R037 |
| Limiter (Lower Bound) | generic package | R038 |
| Absolute limiter | generic package | R160 |
| Absolute limiter with flag | generic package | R202 |
| General First Order Filter | generic package | R109 |
| Tustin Lag Filter | generic package | R162 |
| Tustin Lead-Lag Filter | generic package | R161 |
| Second Order (Notch) Filter | generic package | R110, R111 |
| Tustin Integrator with Limit | generic package | R203 |
| Tustin Integrator with Asymmetric Limit | generic package | |

#### 3.6.8.6.2 INPUT/OUTPUT

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

| Name | Type | Description |
|---|---|---|
| Limit_Relations (Lower Bounds) | enumeration | Establishes the relationship between a signal and the limit imposed on that signal. |

3.6.8.6.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.8.6.4 LOCAL ENTITIES

None.

3.6.8.6.5 INTERRUPTS

None.

3.6.8.6.6 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Signal_Processing, Autopilot_Data_Types;
procedure USER is
   type Command_Signals is Autopilot_Data_Types.Roll_Commands;
   package Command_Limiter is new Signal_Processing.Absolute_Limiter
                                    (Signal_Type => Command_Signals,
                                     Initial_Absolute_Limit => 5.0);
   Command,
   Limited_Signal : Command_Signals;
begin
   Limited_Signal := Command_Limiter.Limit (Command);
   Command_Limiter.Update_Limit (New_Absolute_Limit => 2.5);
end USER;
```

3.6.8.6.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.8.6.8 DECOMPOSITION

Packages:

The following table lists the packages contained in this package and their general description:

| Name | Type | Description |
|---|---|---|
| Upper_Lower_Limiter | generic | Performs limiter function on input signal. Initializes limits and allows updating of limits. |
| Upper_Limiter | generic | Performs limiter function with upper limit only. Initializes limit and allows updating of limit. |
| Lower_Limiter | generic | Performs limiter function with lower limit only. Initializes limit and allows updating of limit. |
| Absolute_Limiter | generic | Performs limiter function based on absolute value of signal. Initializes limit and allows updating. |
| Absolute_Limiter_with_Flag | generic | Performs limiter function based on absolute value of signal and sets flag if upper or lower limit reached. Initializes limit and allows updating. |
| General_First_Order_Filter | generic | Performs first order filter operation according to general method using three coefficients. Also performs initialization of coefficients and allows their values to be updated. |
| Tustin_Lag_Filter | generic | Performs first order filter operation according to Tustin method. Performs initialization of coefficients and allows their values to be updated. |
| Tustin_Lead_Lag_Filter | generic | Performs first order filter operation according to Tustin Lead Lag method. Initializes coefficients and allows their values to be updated. |
| Second_order_Filter | generic | Performs second order filter operation according to Notch filter method. Initializes coefficients and allows their values to be updated. |
| Tustin_integrator_with_limit | generic | Performs integration operation on signal and limits output. |
| Tustin_Integrator_with_Asymmetric_Limit | generic | Performs integration operation on signal and limits output. |

3.6.8.6.9 PART DESIGN

3.6.8.6.9.1 UPPER_LOWER_LIMITER (CATALOG #P71-0)

This package exports operations to perform a limiter function on an input signal (with both upper and lower bounds) and to update the values of the bounds. The package initializes the limits as part of the elaboration of the instantiation.

3.6.8.6.9.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R108.

3.6.8.6.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Signal_Type | generic float | Defines data type for incoming signals to limiter. |

Data Objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Upper_Limit | Signal_Type | Initial value of upper limit signals to limiter. |
| Initial_Lower_Limit | Signal_Type | Initial value of lower limit signals to limiter. |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

| Name | Description |
|---|---|
| Limit_Exception | This exception is raised if the value of the Initial_Upper_Limit <= Initial_Lower_Limit |

3.6.8.6.9.1.3 LOCAL ENTITIES

Data Structures: Must internally store upper and lower limits.

Subprograms:

Must perform initialization of upper and lower limits from generic object parameters, setting Limit_Exception if upper limit not greater than lower limit.

3.6.8.6.9.1.4 INTERRUPTS

None.

3.6.8.6.9.1.5 TIMING AND SEQUENCING

```
with Signal_Processing, Autopilot_Data_Types;
procedure USER is
   type Command_Signals is Autopilot_Data_Types.Roll_Commands;
   package Command_Limiter is new Signal_Processing.Upper_Lower_Limiter
                                    (Signal_Type => Command_Signals,
                                     Initial_Upper_Limit => 5.0,
                                     Initial_Lower_Limit => 0.0);
   Command,
   Limited_Signal : Command_Signals;
begin
   Limited_Signal := Command_Limiter.Limit (Command);
   Command_Limiter.Update_Limits (New_Upper_Limit => 2.5,
                                  New_Lower_Limit => 0.0);
end USER;
```

3.6.8.6.9.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.6.9.1.7 DECOMPOSITION

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Update_Limits | Procedure | Updates values of upper and lower limits |
| Limit | Function | Returns limited value of input signal |

3.6.8.6.9.1.8 PART DESIGN

None.

3.6.8.6.9.2 UPPER_LIMITER (CATALOG #P72-0)

This package exports operations to perform a limiter function on an input signal (with upper bounds) and to update the value of the bounds. The package initializes the limits as part of the elaboration of the instantiation.

3.6.8.6.9.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R037.

3.6.8.6.9.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Signal_Type | generic float | Defines data type for incoming signals to limiter. |

Data Objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Upper_Limit | Signal_Type | Initial value of upper limit signals to limiter. |

3.6.8.6.9.2.3 LOCAL ENTITIES

Data Structures:

Must internally store upper limit.

Subprograms:

Must perform initialization of upper limit from generic object parameter.

3.6.8.6.9.2.4 INTERRUPTS

None.

3.6.8.6.9.2.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Signal_Processing, Autopilot_Data_Types;
procedure USER is
   type Command_Signals is Autopilot_Data_Types.Roll_Commands;
   package Command_Limiter is new Signal_Processing.Upper_Limiter
                                  (Signal_Type => Command_Signals,
                                   Initial_Upper_Limit => 5.0);
```

```
   Command,
   Limited_Signal : Command_Signals;
begin
   Limited_Signal := Command_Limiter.Limit (Command);
   Command_Limiter.Update_Limit (New_Upper_Limit => 2.5);
end USER;
```

3.6.8.6.9.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.6.9.2.7 DECOMPOSITION

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Update_Limit | Procedure | Updates value of upper limit |
| Limit | Function | Returns limited value of input |

3.6.8.6.9.2.8 PART DESIGN

None.

3.6.8.6.9.3 ABSOLUTE_LIMITER (CATALOG #P74-0)

This package exports operations to perform a limiter function on an input signal (with an absolute bound) and to update the value of the bounds. The package initializes the limits as part of the elaboration of the instantiation.

3.6.8.6.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R160.

3.6.8.6.9.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Signal_Type | generic float | Defines data type for incoming signals to limiter. |

Data Objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Absolute Limit | Signal_Type | Initial value of absolute limit |

3.6.8.6.9.3.3 INPUT/OUTPUT

None.

3.6.8.6.9.3.4 LOCAL ENTITIES

Data Structures:

Must internally store absolute limit.

Subprograms:

(1) Must perform initialization of absolute limit from generic object parameter. (2) Must calculate sign of input signal.

3.6.8.6.9.3.5 INTERRUPTS

None.

3.6.8.6.9.3.6 TIMING AND SEQUENCING

```
with Signal_Processing, Autopilot_Data_Types;
procedure USER is
   type Command_Signals is Autopilot_Data_Types.Roll_Commands;
   package Command_Limiter is new Signal_Processing.Absolute_Limiter
                                  (Signal_Type => Command_Signals,
                                   Initial_Absolute_Limit => -5.0);
   Command,
   Limited_Signal : Command_Signals;
begin
   Limited_Signal := Command_Limiter.Limit (Command);
   Command_Limiter.Update_Limit (New_Absolute_Limit => 2.5);
end USER;
```

3.6.8.6.9.3.7 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.6.9.3.8 DECOMPOSITION

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Update_Limit | Procedure | Updates value of absolute limit |
| Limit | Function | Returns limited value of input |

3.6.8.6.9.3.9 PART DESIGN

None.

3.6.8.6.9.4 ABSOLUTE_LIMITER_WITH_FLAG (CATALOG #P75-0)

This package exports operations to perform a limiter function on an input signal (with an absolute bound). The flag will determine if the current input is within the absolute limits, above the upper limit or below the lower limit. The part performs the limit operation, setting or resetting the limit relation as appropriate. It can also update the the value of the limit, and test the value of the flag. The package initializes the limits as part of the elaboration of the instantiation.

3.6.8.6.9.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R202.

3.6.8.6.9.4.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Signal_Type | generic float | Defines data type for incoming signals to limiter. |

Data Objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Absolute_Limit | Signal_Type | Initial value of absolute limit |

3.6.8.6.9.4.3 INPUT/OUTPUT

None.

3.6.8.6.9.4.4 LOCAL ENTITIES

Data Structures:

Must internally store absolute limit.

Subprograms:

(1) Must perform initialization of absolute limit from generic object parameter. (2) Must calculate sign of input signal.

3.6.8.6.9.4.5 INTERRUPTS

None.

3.6.8.6.9.4.6 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Signal_Processing, Autopilot_Data_Types;
procedure USER is
   type Command_Signals is Autopilot_Data_Types.Roll_Commands;
   package Command_Limiter is new
             Signal_Processing.Absolute_Limiter_With_Flag
                (Signal_Type => Command_Signals,
                 initial_Absolute_Limit => -5.0);
   Command,
   Limited_Signal : Command_Signals;
begin
   Limited_Signal := Command_Limiter.Limit (Command);
   Command_Limiter.Update_Limit (New_Absolute_Limit => 2.5);
end USER;
```

3.6.8.6.9.4.7 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.6.9.4.8 DECOMPOSITION

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Update_Limit | Procedure | Updates value of absolute limit |
| Limit_Flag_ Setting | Function | Returns Limit_Relations type giving relation of signal to limit |
| Limit | Function | Returns limited value of input |

3.6.8.6.9.4.9 PART DESIGN

None.

3.6.8.6.9.5 GENERAL_FIRST_ORDER_FILTER (CATALOG #P76-0)

This package exports operations to perform a filter function on an input signal. The part performs the first order fileter operation, and can also update the values of the coefficients to the filter. The package initializes the filter as part of the elaboration of the instantiation.

The form of the filter operations is as follows:

```
X           := (c1 * Input_Signal) + (c2 * Prev_Input) + (c3 * Prev_Output)
Prev_Input  := Input_Signal;
Prev_Output := X;
```

3.6.8.6.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R109.

3.6.8.6.9.5.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Signal_Type | generic float | Defines data type for incoming signals to filter. |
| Coefficient_Type | generic float | Defines data type for incoming coefficients to filter. |

Data Objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Previous_Input_Signal | Signal_Type | Initial value of input signal for first pass |
| Initial_Coefficient_1,_2 | Coefficient_Type | Initial values of coefficients to the filter |

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | Function | Signal_Type * Coefficient_Type return Signal_Type |
| "/" | Function | Signal_Type / Coefficient_Type return Signal_Type |

3.6.8.6.9.5.3 INPUT/OUTPUT

None.

3.6.8.6.9.5.4 LOCAL ENTITIES

Data Structures:

Must internally store coefficients and previous input and output signals

Subprograms:

Must perform initialization of coefficients and previous input signal and calculate value of previous output signal.

3.6.8.6.9.5.5 INTERRUPTS

None.

3.6.8.6.9.5.6 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Signal_Processing, Autopilot_Data_Types;
procedure USER is
   type Command_Signals is Autopilot_Data_Types.Roll_Commands;
   type Coefficients is Autopilot_Data_Types.Degrees_To_Degrees_Gains;
```

```
   package Command_Filter is new
               Signal_Processing.General_First_Order_Filter
                  (Signal_Type                  => Command_Signals,
                   Coefficient_Type             => Coefficients,
                   Initial_Previous_Input_Signal => 0.0,
                   Initial_Coefficient_1        => 0.988,
                   Initial_Coefficient_2        => 0.118,
                   Initial_Coefficient_3        => 0.0988);
   Command,
   Filtered_Signal : Command_Signals;
begin
   Filtered_Signal := Command_Filter.Filter (Command);
   Command_Filter.Update_Coefficients (Coefficient_1 => 0.977,
                                       Coefficient_2 => 0.098,
                                       Coefficient_3 => 0.122);
end USER;
```

3.6.8.6.9.5.7 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.6.9.5.8 DECOMPOSITION

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Update_Coefficients | Procedure | Updates values of the three Coefficients |
| Filter | Function | Returns filtered value of input |

3.6.8.6.9.5.9 PART DESIGN

None.

3.6.8.6.9.6 TUSTIN_LEAD_LAG_FILTER (CATALOG #P77-0)

This package exports operations to perform a filter function on an input signal. The part performs the Tustin Lead Lag filter operation, and can also update the values of the coefficients to the filter. The package initializes the filter as part of the elaboration of the instantiation.

The form of the filter operations is as follows:

```
    X           := (c1 * (Input_Signal - Prev_Input) +
                   (c2 * (Prev_Output - Prev_Input) +   Prev_Input)
    Prev_Input  := Input_Signal;
    Prev_Output := X;
```

3.6.8.6.9.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R161.

3.6.8.6.9.6.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Signal_Type | generic float | Defines data type for incoming signals to filter. |
| Coefficient_Type | generic float | Defines data type for incoming coefficients to filter. |

Data Objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Previous_Input_Signal | Signal_Type | Initial value of input signal for first pass |
| Initial_Coefficient_1,_2 | Coefficient_Type | Initial values of coefficients to the filter |

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | Function | Signal_Type * Coefficient_Type return Signal_Type |

3.6.8.6.9.6.3 INPUT/OUTPUT

None.

3.6.8.6.9.6.4 LOCAL ENTITIES

Data Structures:

Must internally store coefficients and previous input and output signals

Subprograms:

Must perform initialization of coefficients and previous input signal and calculate value of previous output signal.

3.6.8.6.9.6.5 INTERRUPTS

None.

3.6.8.6.9.6.6 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Signal_Processing, Autopilot_Data_Types;
procedure USER is
   type Command_Signals is Autopilot_Data_Types.Roll_Commands;
   type Coefficients is Autopilot_Data_Types.Degrees_To_Degrees_Gains;
   package Command_Filter is new
               Signal_Processing.Tustin_Lead_Lag_Filter
                  (Signal_Type                  => Command_Signals,
                   Coefficient_Type             => Coefficients,
                   Initial_Previous_Input_Signal => 0.0,
                   Initial_Coefficient_1        => 0.988,
                   Initial_Coefficient_2        => 0.0988);
   Command,
   Filtered_Signal : Command_Signals;
begin
   Filtered_Signal := Command_Filter.Filter (Command);
   Command_Filter.Update_Coefficients (New_Coefficient_1 => 0.977,
                                       New_Coefficient_2 => 0.122);
end USER;
```

3.6.8.6.9.6.7 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.6.9.6.8 DECOMPOSITION

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Update_Coefficients | Procedure | Updates values of the two Coefficients |
| Filter | Function | Returns filtered value of input |

3.6.8.6.9.6.9 PART DESIGN

None.

3.6.8.6.9.7 TUSTIN_LAG_FILTER (CATALOG #P78-0)

This package exports operations to perform a filter function on an input signal. The part performs the Tustin Lag filter operation, and can also update the values of the coefficients to the filter. The package initializes the filter as part of the elaboration of the instantiation.

The form of the filter operations is as follows:

```
X           := (c1 * (Input_Signal - Prev_Input) + (c2 * Prev_Output)
Prev_Input  := Input_Signal;
Prev_Output := X;
```

3.6.8.6.9.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R109.

3.6.8.6.9.7.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Signal_Type | generic float | Defines data type for incoming signals to filter. |
| Coefficient_Type | generic float | Defines data type for incoming coefficients to filter. |

Data Objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Previous_Input_Signal | Signal_Type | Initial value of input signal for first pass |
| Initial_Coefficient_1, _2 | Coefficient_Type | Initial values of coefficients to the filter |

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | Function | Signal_Type * Coefficient_Type return Signal_Type |

3.6.8.6.9.7.3 INPUT/OUTPUT

None.

3.6.8.6.9.7.4 LOCAL ENTITIES

Data Structures:

Must internally store coefficients and previous input and output signals

Subprograms:

Must perform initialization of coefficients and previous input signal and calculate value of previous output signal.

3.6.8.6.9.7.5 INTERRUPTS

None.

3.6.8.6.9.7.6 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Signal_Processing, Autopilot_Data_Types;
procedure USER is
   type Command_Signals is Autopilot_Data_Types.Roll_Commands;
   type Coefficients is Autopilot_Data_Types.Degrees_To_Degrees_Gains;
   package Command_Filter is new
               Signal_Processing.Tustin_Lead_Filter
                  (Signal_Type                  => Command_Signals,
                   Coefficient_Type             => Coefficients,
                   Initial_Previous_Input_Signal => 0.0,
                   Initial_Coefficient_1        => 0.988,
```

```
                  Initial_Coefficient_2         => 0.0988);
   Command,
   Filtered_Signal : Command_Signals;
begin
   Filtered_Signal := Command_Limiter.Filter (Command);
   Command_Filter.Update_Coefficients (New_Coefficient_1 => 0.977,
                                       New_Coefficient_2 => 0.122);
end USER;
```

3.6.8.6.9.7.7 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.6.9.7.8 DECOMPOSITION

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Update_Coefficients | Procedure | Updates values of the two Coefficients |
| Filter | Function | Returns filtered value of input |

3.6.8.6.9.7.9 PART DESIGN

None.

3.6.8.6.9.8 SECOND_ORDER_FILTER (CATALOG #P79-0)

This package exports operations to perform a filter function on an input signal. The part performs the Second Order filter operation, and can also update the values of the coefficients to the filter through a redefine operation. The package initializes the filter as part of the elaboration of the instantiation.

The form of the filter operations is as follows:

```
   X               := (c1 * (Input_Signal - 2nd_Prev_Input) +
                       (c2 * (Prev_Input - Prev_Output) ) -
                       (c3 * 2nd_Prev_Output);
   2nd_Prev_Input  := Prev_Input;
   Prev_Input      := Input_Signal;
   2nd_Prev_Output := Prev_Output;
   Prev_Output     := X;
```

3.6.8.6.9.8.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R110.

3.6.8.6.9.8.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Signal_Type | generic float | Defines data type for incoming signals to filter. |
| Coefficient_Type | generic float | Defines data type for incoming coefficients defining parameters |

Data Objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Previous_Input_Signal | Signal_Type | Initial value of input signal for first pass |
| Initial_Coefficient_Defining_Parameters | Coefficient_Type | Initial values used in defining the filter coefficients. |

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | Function | Signal_Type * Coefficient_Type return Signal_Type |

3.6.8.6.9.8.3 INPUT/OUTPUT

None.

3.6.8.6.9.8.4 LOCAL ENTITIES

Data Structures:

Must internally store coefficients and previous input and output signals

Subprograms:

Must perform initialization of coefficients and previous input signal and calculate value of previous output signal.

3.6.8.6.9.8.5 INTERRUPTS

None.

3.6.8.6.9.8.6 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Signal_Processing, Autopilot_Data_Types;
procedure USER is
   type Command_Signals is Autopilot_Data_Types.Roll_Commands;
   type Coefficients is Autopilot_Data_Types.Degrees_To_Degrees_Gains;
   package Command_Filter is new
             Signal_Processing.Second_Order_Filter
                (Signal_Type                    => Command_Signals,
                 Coefficient_Type               => Coefficients,
                 Initial_Previous_Input_Signal  => 0.0,
                 Initial_Coefficient_Defining_Parameter_1
                                                => 0.988,
                 Initial_Coefficient_Defining_Parameter_2
                                                => 0.0988);
   Command,
   Filtered_Signal : Command_Signals;
begin
   Filtered_Signal := Command_Filter.Filter (Command);
   Command_Filter.Redefine_Coefficients
      (New_Coefficient_Defining_Parameter_1 => 0.977,
       New_Coefficient_Defining_Parameter_2 => 0.122);
end USER;
```

3.6.8.6.9.8.7 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.6.9.8.8 DECOMPOSITION

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Redefine_ Coefficients | Procedure | Performs procedure on defining parameters to generate new coefficients. |
| Filter | Function | Returns filtered value of input |

3.6.8.6.9.8.9 PART DESIGN

None.

3.6.8.6.9.9 TUSTIN_INTEGRATOR_WITH_LIMIT (CATALOG #P80-0)

This package exports operations to perform Tustin Integrator with Limit operation on successive input signals. The package also provides the ability of updating the values of the integration constant and limit. The package body uses the Absolute_Limiter_with_Flag package to set and test the limit flag. The package initializes the integrator as part of the elaboration of the instantiation.
The form of the integration will be:

Y = Y_prev + (X + X_prev) * gain * 0.5 * integration_time_interval.

3.6.8.6.9.9.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R203.

3.6.8.6.9.9.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Signals | generic float | Defines data type for incoming signals to integrator. |
| States | generic float | Defines data type for signals output from integrator |
| Gained_Signals | generic float | Defines data type for incoming signal after gain applied |
| Gains | generic float | Defines data type for gains |
| Times | generic float | Defines data type of time interval |

Data Objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Signal_Level | Signals | Initial value of input signal for first pass. |
| Initial_Output_Level | States | Initial values of output signal after first pass. |
| Initial_Output_Limit | States | Initial value of limit on integrator output. |
| Initial_Time_Inteval | Times | Initial value of time interval for integration |
| Initial_Tustin_Gain | Gains | Initial value of gain used Tustin integration |

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | Function | Signals * Gains return Gained_Signals |
| "*" | Function | Gained_Signals * Times return States |

3.6.8.6.9.9.3 INPUT/OUTPUT

None.

3.6.8.6.9.9.4 LOCAL ENTITIES

Data Structures:

Must internally store gains and previous input and output signals

Subprograms:

Must perform limit operation and flag setting as specified in R202 and integrator specified in R124. Must perform initialization of gain, previous input signal, previous output signal, and limit.
Packages:
Must implement a local integrate and limit function. Uses the Absolute_Limit_With_Flag package for limit, and the General_Math.Integrator package for these operations.

3.6.8.6.9.9.5 INTERRUPTS

None.

3.6.8.6.9.9.6 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Signal_Processing, Autopilot_Data_Types;
procedure USER is
   type Command_Signals is new Autopilot_Data_Types.Roll_Commands;
   type Command_Gains is new
      Autopilot_Data_Types.Degrees_To_Degrees_Per_Second_Gains;
   type Gained_Command_Signals is new
      Autopilot_Data_Types.Feedback_Rates_Degrees;
   package Command_Integrator is new
             Signal_Processing.Tustin_Integrator_With_Limit
                (Signal_Type           => Command_Signals,
                 Gains                 => Command_Gains,
                 Gained_Signals        => Gained_Command_Signals,
                 Time_Type             => Seconds,
                 Output_Type           => Command_Signals,
                 Initial_Tustin_Gain   => 0.0,
                 Initial_Signal_Level  => 0.0,
                 Initial_Output_Level  => 0.0,
                 Initial_Time_Interval => 1.0/64.0,
                 Initial_Output_Limit  => 5.0);
   Command           : Command_Signals;
   Integrated_Signal : Command_Signals;
begin
   Integrated_Signal := Command_Integrator.Integrate (Command);
   Command_Integrator.Update_Integration_Coefficient
      (New_Absolute_Limit => 2.5);
end USER;
```

3.6.8.6.9.9.7 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.6.9.9.8 DECOMPOSITION

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Update_Limit | Procedure | Updates absolute limit on integrator output. |
| Update_Gain | Procedure | Updates gain on input signal |
| Integrate | Function | Returns integrated value of input. |
| Reset | Procedure | Resets integrator state and previous input to new values |
| Limit_Flag_Setting | Function | Returns Limit_Relations type giving relation of signal to limit |

3.6.8.6.9.9 PART DESIGN

None.

3.6.8.6.9.10 TUSTIN_INTEGRATOR_WITH_ASYMMETRIC_LIMIT (CATALOG #P1053-0)

This package exports operations to perform Tustin Integrator with Limit operation on successive input signals. The package also provides the ability of updating the values of the integration constant and limit. The package body uses the Absolute_Limiter_with_Flag package to set and test the limit flag. The package initializes the integrator as part of the elaboration of the instantiation.
The form of the integration will be:

Y = Y_prev + (X + X_prev) * gain * 0.5 * integration_time_interval.

3.6.8.6.9.10.1 REQUIREMENTS ALLOCATION

None.

3.6.8.6.9.10.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data Types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Signals | generic float | Defines data type for incoming signals to integrator. |
| States | generic float | Defines data type for signals output from integrator |
| Gained_Signals | generic float | Defines data type for incoming signal after gain applied |
| Gains | generic float | Defines data type for gains |
| Times | generic float | Defines data type of time interval |

Data Objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Signal_Level | Signals | Initial value of input signal for first pass. |
| Initial_Output_Level | States | Initial values of output signal after first pass. |
| Initial_Output_Lower_Limit | States | Initial value of lower limit on integrator output. |
| Initial_Output_Upper_Limit | States | Initial value of upper limit on integrator output. |
| Initial_Time_Inteval | Times | Initial value of time interval for integration |
| Initial_Tustin_Gain | Gains | Initial value of gain used Tustin integration |

Subprograms:

The following table shows the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | Function | Signals * Gains return Gained_Signals |
| "*" | Function | Gained_Signals * Times return States |

3.6.8.6.9.10.3 INPUT/OUTPUT

None.

3.6.8.6.9.10.4 LOCAL ENTITIES

Data Structures:

Must internally store gains and previous input and output signals

Subprograms:

Must perform limit operation and flag setting as specified in R108 and integrator specified in R124. Must perform initialization of gain, previous input signal, previous output signal, and limit.
Packages:
Must implement a local integrate and limit function. Uses the Absolute_Limit_With_Flag package for limit, and the General_Math.Integrator package for these operations.

3.6.8.6.9.10.5 INTERRUPTS

None.

3.6.8.6.9.10.6 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Signal_Processing, Autopilot_Data_Types;
procedure USER is
   type Command_Signals is new Autopilot_Data_Types.Roll_Commands;
   type Command_Gains is new
      Autopilot_Data_Types.Degrees_To_Degrees_Per_Second_Gains;
   type Gained_Command_Signals is new
      Autopilot_Data_Types.Feedback_Rates_Degrees;
   package Command_Integrator is new
            Signal_Processing.Tustin_Integrator_With_Assymetric_Limit
               (Signal_Type                => Command_Signals,
                Gains                      => Command_Gains,
                Gained_Signals             => Gained_Command_Signals,
                Time_Type                  => Seconds,
                Output_Type                => Command_Signals,
                Initial_Tustin_Gain        => 0.0,
                Initial_Signal_Level       => 0.0,
                Initial_Output_Level       => 0.0,
                Initial_Time_Interval      => 1.0/64.0,
                Initial_Output_Lower_Limit => -5.0);
                Initial_Output_Upper_Limit => 4.0);
   Command           : Command_Signals;
   Integrated_Signal : Command_Signals;
begin
   Integrated_Signal := Command_Integrator.Integrate (Command);
   Command_Integrator.Update_Integration_Coefficient
      (New_Lower_Limit => 2.5);
end USER;
```

3.6.8.6.9.10.7 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.6.9.10.8 DECOMPOSITION

Subprograms:

The following table shows the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| Update_Limits | Procedure | Updates lower and uppper limits on integrator output. |
| Update_Upper_Limit | Procedure | Updates upper limit on integrator output. |
| Update_Gain | Procedure | Updates gain on input signal |
| Integrate | Function | Returns integrated value of input. |
| Reset | Procedure | Resets integrator state and previous input to new values |

3.6.8.6.9.10.9 PART DESIGN

None.

(This page left intentionally blank.)

```
package Signal_Processing is

---------------------
--Exported Data Type-
---------------------

--Purpose:
--      This data type will be used in the Absolute Limiter and Tustin Integrator
--      Packages.  An object of this type will be set to indicate if a signal
--      coming into the limiter is between the upper and lower limits
--      (WITHIN_LIMIT), above the upper limit (AT_POSITIVE_LIMIT), or below the
--      lower limit (AT_NEGATIVE_LIMIT).

--Requirements trace:
--      This type meets requirements for parts R202 (SRS 3.4.5.7.11 (b))

   type Limit_Relations is (Within_Limit, At_Positive_Limit, At_Negative_Limit);

pragma PAGE;
    generic
        type Signal_Type is digits <>;
        Initial_Upper_Limit : in Signal_Type;
        Initial_Lower_Limit : in Signal_Type;
    package Upper_Lower_Limiter is

        procedure Update_Limits (New_Upper_Limit : in Signal_Type;
                                 New_Lower_Limit : in Signal_Type);

        function Limit (Signal : Signal_Type) return Signal_Type;

        Limit_Exception:  exception;

    end Upper_Lower_Limiter;

pragma PAGE;
    generic
        type Signal_Type is digits <>;
        Initial_Upper_Limit : in Signal_Type;
    package Upper_Limiter is

        procedure Update_Limit (New_Upper_Limit : in Signal_Type);

        function Limit (Signal : Signal_Type) return Signal_Type;

    end Upper_Limiter;

pragma PAGE;
    generic
        type Signal_Type is digits <>;
        Initial_Lower_Limit : in Signal_Type;
    package Lower_Limiter is

        procedure Update_Limit (New_Lower_Limit : in Signal_Type);

        function Limit (Signal : Signal_Type) return Signal_Type;

    end Lower_Limiter;
```

```
pragma PAGE;
    generic
        type Signal_Type is digits <>;
        Initial_Absolute_Limit : in Signal_Type;
    package Absolute_Limiter is

        procedure Update_Limit (New_Absolute_Limit : in Signal_Type);

        function Limit (Signal : Signal_Type) return Signal_Type;

    end Absolute_Limiter;


pragma PAGE;
    generic
        type Signal_Type is digits <>;
        Initial_Absolute_Limit : in Signal_Type;
    package Absolute_Limiter_With_Flag is

        procedure Update_Limit (New_Absolute_Limit : in Signal_Type);

        function Limit_Flag_Setting return Limit_Relations;

        function Limit (Signal : Signal_Type) return Signal_Type;

    end Absolute_Limiter_With_Flag;


pragma PAGE;
    generic
        type Signal_Type is digits <>;
        type Coefficient_Type is digits <>;
        Initial_Previous_Input_Signal : in Signal_Type;
        Initial_Coefficient_1 : in Coefficient_Type;
        Initial_Coefficient_2 : in Coefficient_Type;
        Initial_Coefficient_3 : in Coefficient_Type;
        with function "*" (Left : Signal_Type; Right : Coefficient_Type)
            return Signal_Type is <>;
        with function "/" (Left : in Signal_Type; Right : in Coefficient_Type)
            return Signal_Type is <>;
    package General_First_Order_Filter is

        procedure Update_Coefficients (New_Coefficient_1 : in Coefficient_Type;
                                       New_Coefficient_2 : in Coefficient_Type;
                                       New_Coefficient_3 : in Coefficient_Type);

        function Filter (Signal : Signal_Type) return Signal_Type;

    end General_First_Order_Filter;

pragma PAGE;
    generic
        type Signal_Type is digits <>;
        type Coefficient_Type is digits <>;
        Initial_Previous_Input_Signal : in Signal_Type;
```

```
        Initial_Coefficient_1 : in Coefficient_Type;
        Initial_Coefficient_2 : in Coefficient_Type;
        with function "*" (Left : Signal_Type; Right : Coefficient_Type)
            return Signal_Type is <>;
    package Tustin_Lead_Lag_Filter is

        procedure Update_Coefficients (New_Coefficient_1 : in Coefficient_Type;
                                       New_Coefficient_2 : in Coefficient_Type);

        function Filter (Signal : Signal_Type) return Signal_Type;

    end Tustin_Lead_Lag_Filter;


pragma PAGE;
    generic
        type Signal_Type is digits <>;
        type Coefficient_Type is digits <>;
        Initial_Previous_Input_Signal : in Signal_Type;
        Initial_Coefficient_1 : in Coefficient_Type;
        Initial_Coefficient_2 : in Coefficient_Type;
        with function "*" (Left : Signal_Type; Right : Coefficient_Type)
            return Signal_Type is <>;
    package Tustin_Lag_Filter is

        procedure Update_Coefficients (New_Coefficient_1 : in Coefficient_Type;
                                       New_Coefficient_2 : in Coefficient_Type);

        function Filter (Signal : Signal_Type) return Signal_Type;

    end Tustin_Lag_Filter;


pragma PAGE;
    generic
        type Signal_Type is digits <>;
        type Coefficient_Type is digits <>;
        Initial_Previous_Input_Signal : in Signal_Type;
        Initial_Coefficient_Defining_Parameter_1: in Coefficient_Type;
        Initial_Coefficient_Defining_Parameter_2: in Coefficient_Type;
        with function "*" (Left : Signal_Type; Right : Coefficient_Type)
            return Signal_Type is <>;
    package Second_Order_Filter is

        procedure Redefine_Coefficients
                  (New_Coefficient_Defining_Parameter_1 : in Coefficient_Type;
                   New_Coefficient_Defining_Parameter_2 : in Coefficient_Type);


        function Filter (Signal : Signal_Type) return Signal_Type;

    end Second_Order_Filter;


pragma PAGE;
   generic
      type Signals         is digits <>;
```

```
      type States          is digits <>;
      type Gained_Signals  is digits <>;
      type Gains           is digits <>;
      Initial_Tustin_Gain  : in Gains;
      Initial_Signal_Level : in Signals;
      Initial_State        : in States := 0.0;
      Initial_Signal_Limit : in States;
      with function "*" (Left  : Signals;
                         Right : Gains) return Gained_Signals is <>;
      with function "*" (Left : Gained_Signals;
                         Right : States) return States is <>;
   package Tustin_Integrator_With_Limit is

      procedure Update_Limit (New_Absolute_Limit : in States);

      procedure Update_Gain (New_Gain : in Gains);

      function Integrate (Signal : Signals) return States;

      procedure RESET (Integrator_State : in States;
                       Signal           : in Signals);

      function Limit_Flag_Setting return Limit_Relations;

   end Tustin_Integrator_With_Limit;

pragma PAGE;
   generic
      type Signals         is digits <>;
      type States          is digits <>;
      type Gained_Signals  is digits <>;
      type Gains           is digits <>;

      Initial_Tustin_Gain        : in Gains;
      Initial_Signal_Level       : in Signals;
      Initial_State              : in States := 0.0;
      Initial_Signal_Lower_Limit : in States;
      Initial_Signal_Upper_Limit : in States;

      with function "*" (Left  : Signals;
                         Right : Gains) return Gained_Signals is <>;
      with function "*" (Left  : Gained_Signals;
                         Right : States) return States is <>;

   package Tustin_Integrator_With_Asymmetric_Limit is

      procedure Update_Limits (New_Lower_Limit : in States;
                               New_Upper_Limit : in States);

      procedure Update_Gain (New_Gain : in Gains);

      function Integrate (Signal : Signals) return States;

      procedure RESET (Integrator_State : in States;
                       Signal           : in Signals);

      function Limit_Flag_Setting return Limit_Relations;
```

```
   end Tustin_Integrator_With_Asymmetric_Limit:
end Signal_Processing;
```

(This page left intentionally blank.)

### 3.6.8.7 GENERAL PURPOSE MATH (SPECIFICATION) TLCSC (CATALOG #P11-0)

This TLCSC is a package which consists of two types of subpackages: generic packages and simple packages which contain generic functions. As a group, the subpackages provide the general purpose math routines required by the rest of the CAMP parts.

#### 3.6.8.7.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this

TLCSC:

| Name | Requirements Allocation |
|---|---|
| Lookup_Table_Even_Spacing | R118 |
| Lookup_Table_Uneven_Spacing | R119 |
| Two_Way_Table_Lookup | |
| Incrementor | R120 |
| Decrementor | R121 |
| Running_Average | R142 |
| Change_Calculator | R113 |
| Accumulator | R114 |
| Change_Accumulator | R115 |
| Integrator | R124 |
| Interpolate_or_Extrapolate | R116, R117 |
| Square_Root | R123 |
| Root_Sum_Of_Squares | R122 |
| Sign | R224 |
| Mean_Value | R144 |
| Mean_Absolute_Difference | R143 |

#### 3.6.8.7.2 INPUT/OUTPUT

None.

#### 3.6.8.7.3 UTILIZATION OF OTHER ELEMENTS

None.

#### 3.6.8.7.4 LOCAL ENTITIES

None.

#### 3.6.8.7.5 INTERRUPTS

None.

3.6.8.7.6 TIMING AND SEQUENCING

None.

3.6.8.7.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.8.7.8 DECOMPOSITION

The following table describes the decomposition of this TLCSC:

| Name | Type | Description |
|---|---|---|
| Lookup_Table_ Even_Spacing | generic package | Provides the capability to reinitialize and search through a table of unevenly spaced dependent and independent values |
| Lookup_Table_ Uneven_Spacing | generic package | Provides the capability to initialize and search through a table of evenly spaced independent and dependent value |
| Two_Way_Table_ Lookup | generic package | Provides the capability to initialize and search through a table for either dependent or independent values |
| Incrementor | generic package | Provides the capability to initialize, increment, and read a value. |
| Decrementor | generic package | Provides the capability to initialize, decrement, and read a value. |
| Running_Average | generic package | Provides the capability to maintain a running average. |
| Integrator | generic package | Provides the capability to integrate a variable across time |
| Interpolate_or_ Extrapolate | generic function | Returns value interpolated or extrapolated with two independent values |
| Square Root | generic package | Contains a function which returns the square root of an input value |
| Root_Sum_of_ Squares | generic function | Returns the root sum of three squared values, i.e., Sqrt (X**2 + Y**2 + Z**2) |
| Sign | generic function | Returns -1 if < 0, 1 if >= 0 |
| Mean_Value | generic function | Returns the average value of a vector of numbers |
| Mean_Absolute_ Difference | generic function | Returns average absolute difference between a series of numbers and their average |

3.6.8.7.9 PART DESIGN

3.6.8.7.9.1 LOOKUP_TABLE_EVEN_SPACING (CATALOG #P12-0)

This LLCSC, which is designed as an Ada generic package, provides the ability to initialize and search through a table of independent and dependent values which are evenly spaced. An initialization routine is provided to allow for run-time initialization of the table. However, since the table is exported in the package specification, it may also be initialized at compilation time. A search routine is provided to access sets of data in the table. The search will key on an independent value. If the independent value falls in the range covered by the table, the immediately higher and lower independent values, along with the corresponding dependent values, will be returned. If the independent value falls outside the range covered by the table, the two closest independent values, along with the corresponding dependent values, will be returned.

The exception "Value_Out_Of_Range" is created if Key for Lookup (without flag) is outside of the Table range

3.6.8.7.9.1.1 REQUIREMENTS ALLOCATION

This LLCSC meets CAMP requirement R118.

3.6.8.7.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Base Type | Description |
|---|---|---|
| Dependent_Type | generic float | Type for the dependent variable |
| Independent_Type | generic float | Type for the independent variable |
| Index_Type | discrete | Type for the lookup table index |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Mode | Description |
|---|---|---|---|
| Minimum_Independnt_Value | Independent_Type | in | value of the first independent table value |
| Maximum_Independnt_Value | Independent_Type | in | value of the last independent table value |

Subprograms:

The following table describes the generic formal subprograms required by this LLCSC:

| Name | Type | Description |
|---|---|---|
| "*" | function | Independent_Type := Index_Type * Independent_Type |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following chart describes the exceptions exported by this LLCSC:

| Name | Raised By | Description |
|---|---|---|
| Value_Out_Of_Range | Lookup | The input value has mapped to outside the table range. |

Data types:

The following chart describes the data types exported by this LLCSC:

| Name | Range | Operators | Description |
|---|---|---|---|
| Key_Range_Flag | N/A | None | Specifies whether the req. key is in table range |

3.6.8.7.9.1.3 LOCAL ENTITIES

None.

3.6.8.7.9.1.4 INTERRUPTS

None.

3.6.8.7.9.1.5 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Test is

   type Dep_Type is digits 6;
   type Ind_Type is digits 6;
   type Index    is range 1 .. 3;

   Lower_Dep_Value  : Dep_Type;
```

```
      Higher_Dep_Value : Dep_Type;
      Lower_Ind_Value  : Ind_Type;
      Higher_Ind_Value : Ind_Type;

      package Table is new General_Purpose_Math.Lookup_Table_Even_Spacing
                 (Dependent_Type           => Dep_Type,
                  Independent_Type         => Ind_Type,
                  Table_Range              => Index,
                  Minimum_Independent_Value => 10.0,
                  Maximum_Independent_Value => 30.0);
   begin
      Table.Initialize (Index => 1, Dependent_Value => 20.0);
      Table.Initialize (Index => 2, Dependent_Value => 50.0);
      Table.Initialize (Index => 3, Dependent_Value => 90.0);

      Table.Lookup (Key => 45.0,
                    Lower_Independent => Lower_Ind,
                    Higher_Independent => Higher_Ind,
                    Lower_Dependent    => Lower_Dep,
                    Higher_Dependent   => Higher_Dep);
   end Test;
```

#### 3.6.8.7.9.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

#### 3.6.8.7.9.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Initialize | procedure | Initialize one row of the table |
| Lookup | procedure | Do a table lookup (raise exception if key is outside the table range) |
| Lookup | procedure | Do a table lookup (return flag specifying whether key is in table range) |

#### 3.6.8.7.9.1.8 PART DESIGN

None.

### 3.6.8.7.9.2 LOOKUP_TABLE_UNEVEN_SPACING (CATALOG #P13-0)

This LLCSC, which is designed as an Ada generic package, provides the ability to initialize and search through a table of independent and dependent values which are unevenly spaced. An initialization routine is provided to allow for run-time initialization of the table. However, since the table is exported in the package specification, it may also be initialized at compilation time. A search routine is provided to access sets of data in the table. The search will key on an independent value. If the independent value falls in the range

covered by the table, the immediately higher and lower independent values, along with the corresponding dependent values, will be returned. If the independent value falls outside the range covered by the table, the two closest independent values, along with the corresponding dependent values, will be returned.

The exception "Value_Out_Of_Range" is created if Key for Lookup (without flag) is outside of the Table range

3.6.8.7.9.2.1 REQUIREMENTS ALLOCATION

This LLCSC meets CAMP requirement R119.

3.6.8.7.9.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Base Type | Description |
|---|---|---|
| Independent_Type | gen. float | Type for the independent variable |
| Dependent_Type | gen. float | Type for the dependent variable |
| Index_Type | discrete | Type for the table index |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following chart describes the exceptions exported by this LLCSC:

| Name | Raised By | Description |
|---|---|---|
| Value_Out_Of_ Range | Lookup | The input value has mapped to outside the table range. |

Data types:

The following chart describes the data types exported by this LLCSC:

| Name | Range | Operators | Description |
|---|---|---|---|
| Key_Range Flag | N/A | None | Specifies whether the req. key is in table range |
| Table_ Entries | N/A | None | Record describing the makeup of one table entry |

Data objects:

The following table describes the data objects exported by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Table | array | N/A | This is the lookup table created |

3.6.8.7.9.2.3 LOCAL ENTITIES

None.

3.6.8.7.9.2.4 INTERRUPTS

None.

3.6.8.7.9.2.5 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is

   type Dep_Type is digits 6;
   type Ind_Type is digits 6;
   type Index    is range 1 .. 3;

   Lower_Dep_Value  : Dep_Type;
   Higher_Dep_Value : Dep_Type;
   Lower_Ind_Value  : Ind_Type;
   Higher_Ind_Value : Ind_Type;

   package Table is new General_Purpose_Math.Lookup_Table_Uneven_Spacing
              (Dependent_Type            => Dep_Type,
               Independent_Type          => Ind_Type,
               Table_Range               => Index);
begin
   Table.Initialize (Index => 1, Independent_Value => 10, Dependent_Value => 1);
   Table.Initialize (Index => 2, Independent_Value => 15, Dependent_Value => 2);
   Table.Initialize (Index => 3, Independent_Value => 25, Dependent_Value => 3);

   Table.Lookup (Key => 17,
                 Lower_Independent => Lower_Ind,
                 Higher_Independent => Higher_Ind,
                 Lower_Dependent    => Lower_Dep,
                 Higher_Dependent   => Higher_Dep);
end Test;
```

3.6.8.7.9.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.7.9.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Initialize | procedure | Initialize one row of the table |
| Lookup | procedure | Do a table lookup (raise exception if key is outside the table range) |
| Lookup | procedure | Do a table lookup (return flag specifying whether key is in table range) |

3.6.8.7.9.2.8 PART DESIGN

None.

3.6.8.7.9.3 INCREMENTOR (CATALOG P14-0)

This generic package provides the capability to reinitialize a variable that is to be incremented, select a value to be used as an incrementor, and increment the variable accordingly. A reinitialization routine is provided to reinitialize the variable and the increment amount. An increment routine is provided to do the actual incrementing.

3.6.8.7.9.3.1 REQUIREMENTS ALLOCATION

This LLCSC meets CAMP requirement R120.

3.6.8.7.9.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Base Type | Description |
|---|---|---|
| Real_Type | gen. float | Type of the incrementor variable |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Mode | Description |
|---|---|---|---|
| Initial_Value | Real_Type | in | Initial incrementor value |
| Increment_Amount | Real_Type | in | Amount by which to increment |

3.6.8.7.9.3.3 LOCAL ENTITIES

None.

3.6.8.7.9.3.4 INTERRUPTS

None.

3.6.8.7.9.3.5 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is
   type Message_Type is digits 6;
   Number_Of_Messages : Message_Type;

   package Message is new General_Purpose_Math.Incrementor
                          (Real_Type       => Message_Type,
                           Initial_Value   => 2.0,
                           Increment_Amount => 1.0);
begin
   Number_Of_Messages := Message.Increment;
end Sample;
```

3.6.8.7.9.3.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.7.9.3.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Reinitialize | procedure | Resets the incrementor value and increment amount |
| Increment | function | Increments the variable and returns its new value |

3.6.8.7.9.3.8 PART DESIGN

None.

3.6.8.7.9.4 DECREMENTOR (CATALOG #P15-0)

This generic package provides the capability to reinitialize a variable that is to be decremented, select a value to be used as an decrementor, and decrement the variable accordingly. A reinitialization routine is provided to reinitialize the variable and the decrement amount. An decrement routine is provided to do the actual decrementing.

3.6.8.7.9.4.1 REQUIREMENTS ALLOCATION

This LLCSC meets CAMP requirement R121.

3.6.8.7.9.4.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Base Type | Description |
|---|---|---|
| Real_Type | generic float | Type of the decrementor variable |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Mode | Description |
|---|---|---|---|
| Initial_Value | Real_Type | in | Initial decrementor value |
| Decrement_Amount | Real_Type | in | Amount by which to decrement |

3.6.8.7.9.4.3 LOCAL ENTITIES

None.

3.6.8.7.9.4.4 INTERRUPTS

None.

3.6.8.7.9.4.5 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is
   type Message_Type is digits 6;
   Number_Of_Messages : Message_Type;

   package Message is new General_Purpose_Math.Decrementor
                          (Real_Type       => Message_Type,
                           Initial_Value   => 2.0,
                           Increment_Amount => 1.0);
begin
   Number_Of_Messages := Message.Decrement;
end Sample;
```

3.6.8.7.9.4.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.7.9.4.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Reinitialize | procedure | Resets the decrementor value and decrement amount |
| Decrement | function | Decrements the variable and returns its new value |

3.6.8.7.9.4.8 PART DESIGN

None.

3.6.8.7.9.5 RUNNING_AVERAGE (CATALOG #P16-0)

This generic package provides the capability to initialize a sum and/or a count and to maintain a running average. A reinitialization routine is provided to reinitialize the sum and count. An averaging routine is provided to perform the running sum.

3.6.8.7.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R142

3.6.8.7.9.5.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Base Type | Description |
|---|---|---|
| Real_Type | gen. float | Type of the running average var. |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Mode | Description |
|---|---|---|---|
| Initial_Sum | Real_Type | in | Initial running sum |
| Initial_Count | INTEGER | in | Initial # of data points |

Subprograms:

The following table describes the generic formal subprograms required by this LLCSC:

| Name | Type | Description |
|---|---|---|
| "/" | function | Real_Type := Real_Type / Integer |

3 6.8.7.9.5.3 LOCAL ENTITIES

None.

3.6.8.7.9.5.4 INTERRUPTS

None.

3.6.8.7.9.5.5 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is
   type Test_Type is digits 6;
   New_Average : Test_Type;

   package Test is new General_Purpose_Math.Running_Average
```

```
                                (Real_Type     => Test_Type,
                                 Initial_Sum   => 25.0,
                                 Initial_Count => 5);
   begin
      New_Average := Test.Current_Average (New_Value => 11.0);
   end Sample;
```

3.6.8.7.9.5.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.7.9.5.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Reinitialize | procedure | Sets up initial sum, and count |
| Reinitialize | procedure | Sets up initial count |
| Current_Average | function | Given new value, returns new average |

3.6.8.7.9.5.8 PART DESIGN

None.

3.6.8.7.9.6 ACCUMULATOR (CATALOG #P17-0)

This generic package provides a set of operations for maintaining an accumulation of a subject variable.

3.6.8.7.9.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R114.

3.6.8.7.9.6.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Base Type | Description |
|---|---|---|
| Element_Type | generic float | Type of the variable being accumulated. |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Mode | Description |
|---|---|---|---|
| Initial_Value | Real_Type | in | Initial accumulator value |

3.6.8.7.9.6.3 LOCAL ENTITIES

None.

3.6.8.7.9.6.4 INTERRUPTS

None.

3.6.8.7.9.6.5 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is
   type Test_Type is digits 6;
   Accumulator_Value : Test_Type;

   package Test is new General_Purpose_Math.Accumulator
                              (Element_Type  => Test_Type,
                               Initial_Value => 25.0);
begin
   Test.Accumulate (New_Value => 10.0);
   Accumulator_Value := Test.Retrieve;
end Sample;
```

3.6.8.7.9.6.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.7.9.6.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Reinitialize | procedure | Give initial value for accumulated var. |
| Accumulate | procedure | Add to current value of tracked variable |
| Accumulate | procedure | Add to current accumulated value and return new value |
| Retrieve | function | Retrieve current accumulated value |

3.6.8.7.9.6.8 PART DESIGN

None.

3.6.8.7.9.7 CHANGE_CALCULATOR (CATALOG #P18-0)

This generic package provides a set of operations for tracking the change in a given variable.

3.6.8.7.9.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R113.

3.6.8.7.9.7.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Base Type | Description |
|---|---|---|
| Element_Type | generic float | Type of the variable being trackd |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Mode | Description |
|---|---|---|---|
| Initial_PV | Element_Typ | in | Initial previous value |

3.6.8.7.9.7.3 LOCAL ENTITIES

None.

3.6.8.7.9.7.4 INTERRUPTS

None.

3.6.8.7.9.7.5 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is
```

```
      type Test_Type is digits 6;
      Change_Value : Test_Type;

      package Test is new General_Purpose_Math.Change_Calculator
                              (Element_Type  => Test_Type,
                               Initial_Value => 25.0);
   begin
      Change_Value := Test.Change (New_Value => 40.0);
   end Sample;
```

3.6.8.7.9.7.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.7.9.7.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Reinitialize | procedure | Reininitalize value of tracked variable |
| Change | function | Return change since the last call |
| Retrieve_Value | function | Return current value of tracked variable |

3.6.8.7.9.7.8 PART DESIGN

None.

3.6.8.7.9.8 CHANGE_ACCUMULATOR (CATALOG #P19-0)

This generic package provides a set of operations for maintaining an accumulation of a change to a subject variable.

3.6.8.7.9.8.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R115.

3.6.8.7.9.8.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Base Type | Description |
|---|---|---|
| Element_Type | generic float | Type of the variable being tracked and accumulated. |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Mode | Description |
|---|---|---|---|
| Initial_PV | Element_Typ | in | Initial previous value |
| Initial_Accumulator_Value | Element_Typ | in | Initial accumulator value |

3.6.8.7.9.8.3 LOCAL ENTITIES

None.

3.6.8.7.9.8.4 INTERRUPTS

None.

3.6.8.7.9.8.5 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is
   type Test_Type is digits 6;
   Change_Value      : Test_Type;

   package Test is new General_Purpose_Math.Change_Accumulator
                              (Element_Type              => Test_Type,
                               Initial_Previous_Value    => 25.0,
                               Initial_Accumulator_Value => 0.0);
begin
   Test.Accumulate (New_Value => 15.0);
   Change_Value := Test.Retrieve_Previous_Value;
end Sample;
```

3.6.8.7.9.8.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.7.9.8.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Reinitialize | procedure | Give initial value for accumulator variable |
| Reinitialize | procedure | Give initial value for accumulator and previous value variables |
| Accumulate_Change | procedure | Accumulate the change in the variable |
| Accumulate_Change | procedure | Accumulate the change in the variable and return new value |
| Retrieve_Accumulator | function | Return current accumulator value |
| Return_Previous_Value | function | Return current value of previous value variable |

3.6.8.7.9.8.8 PART DESIGN

None.

3.6.8.7.9.9 INTEGRATOR (CATALOG #P20-0)

This generic package manages a data value and allows it to be integrated across time by means of a trapezoidal integration technique.

3.6.8.7.9.9.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R124.

3.6.8.7.9.9.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Base Type | Description |
|---|---|---|
| Dependent_Type | generic float | Type of the dependent variable |
| Independent_Type | generic float | Type of the independent variable |
| Time_Interval | generic float | Type of the delta time variable |

Data objects:

The following table describes the genric formal object required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Independent_Value | Independent_Type | Initial value for independent variable |
| Initial_Dependent_Value | Dependent_Type | Initial value for dependent variable |
| Default_Delta_Time | Time_Interval | Default time between integration |

Subprograms:

The following table describes the generic formal subprograms required by this LLCSC:

| Name | Type | Description |
|---|---|---|
| "*" | function | Dependent_Type := Independent_Type * Time_Interval |

### 3.6.8.7.9.9.3 LOCAL ENTITIES

None.

### 3.6.8.7.9.9.4 INTERRUPTS

None.

### 3.6.8.7.9.9.5 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is
   type Dependent   is digits 6;
   type Independent is digits 6;
   New_Dependent_Value : Dependent;

   package Test is new General_Purpose_Math.Integrator
                        (Dependent_Type   => Dependent,
                         Independent_Type => Independent,
                         Initial_Dependent_Value   => 25.0,
                         Initial_Independent_Value => 10.0,
                         Default_Delta_Time        =>  0.1);
begin
   New_Dependent_Value := Test    .egrate (Current_Independent_Value => 30.0);
end Sample;
```

11

3.6.8.7.9.9.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.7.9.9.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Reinitialize | procedure | Give initial dependent and independent values |
| Update | procedure | Give new value for independent value |
| Integrate | function | Integrate across time |

3.6.8.7.9.9.8 PART DESIGN

None.

3.6.8.7.9.10 INTERPOLATE OR EXTRAPOLATE (CATALOG #P21-0)

This part is a generic function which computes the linear interpolation between two values.

3.6.8.7.9.10.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirements R116 and R117

3.6.8.7.9.10.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this unit:

| Name | Base Type | Description |
|---|---|---|
| Independent_Type | gener. float | Type of the independent variables |
| Dependent_Type | gener. float | Type of the dependent variable |
| Dependent_over_Independent_Type | gener. float | Result of Dependent / Independent |

Subprograms:

The following table describes the generic formal subprograms required by this unit:

| Name | Type | Description |
|---|---|---|
| "/" | function | Dependent_Over_Independent_Type := Dependent_Type / Independent_Type |
| "*" | function | Dependent_Type := Dependent_Over_ Independent_Type * Independent_Type |

FORMAL PARAMETERS:

The following table describes this unit's formal parameters:

| Name | Type | Mode | Description |
|---|---|---|---|
| Input | Independent | in | Independent value for which a dependent value is returned |
| Lower_Independent | Independent | in | Lower independent value |
| Higher_Independent | Independent | in | Higher independent value |
| Lower_Dependent | Dependent | in | Lower dependent value |
| Higher_Dependent | Dependent | in | Higher dependent value |
| <return value> | Dependent | out | Computed interpolated value |

3.6.8.7.9.10.3 INTERRUPTS

None.

3.6.8.7.9.10.4 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is
   type Dependent   is digits 6;
   type Independent is digits 6;
   Key,     Lower_Ind, Higher_Ind : Independent;
   New_Dep, Lower_Dep, Higher_Dep : Dependent;

   function Interp_or_Extrap is new
                General_Purpose_Math.Interpolate_or_Extrapolate
                     (Dependent_Type   => Dependent,
                      Independent_Type => Independent);
begin
   New_Dep := Interp_or_Extrap
                    (Input             => Key,
                     Lower_Dependent   => Lower_Dep,
                     Higher_Dependent  => Higher_Dep,
                     Lower_Independent => Lower_Ind,
                     Higher_Independent => Higher_Ind);
end Sample;
```

3.6.8.7.9.10.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.7.9.10.6 DECOMPOSITION

N/A

3.6.8.7.9.11 SQUARE_ROOT (CATALOG #P23-0)

This part is a generic package which computes the square root of an input value.

The Ada predefined exception "Numeric_Error" is raised if "Input" is negative.

3.6.8.7.9.11.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R123.

3.6.8.7.9.11.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this unit:

| Name | Type | Description |
|---|---|---|
| Inputs | floating point | Data type of input values |
| Outputs | floating point | Data type of output values |
| Real | floating point | Unconstrained type for intermediate calculations |

FORMAL PARAMETERS:

The following table describes this unit's formal parameters:

| Name | Type | Mode | Description |
|---|---|---|---|
| Input | Squared_Type | in | Input value to square root operation |
| <return value> | Real_Type | out | Result of square root operation |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following exceptions are exported by this part:

| Name | Description |
|---|---|
| Negative_Input | Input to the Square Root function was negative |

3.6.8.7.9.11.3 LOCAL ENTITIES

Packages:

The body of this part instantiates the Square Root function in the Polynomials package.

3.6.8.7.9.11.4 INTERRUPTS

None.

3.6.8.7.9.11.5 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is
   type Sin_Cos_Ratio is digits 6 range -1.0..1.0;
   type Real          is digits 6;
   Result : Sin_Cos_Ratio;
   Input  : Sin_Cos_Ratio;

   package My is new General_Purpose_Math.Square_Root
                        (Inputs  => Sin_Cos_Ratio,
                         Outputs => Sin_Cos_Ratio,
                         Real    => Real);
begin
   Result := My.Sqrt (Input => Input);
end Sample;
```

3.6.8.7.9.11.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.7.9.11.7 DECOMPOSITION

None.

3.6.8.7.9.11.8 PART DESIGN

None.

3.6.8.7.9.12 ROOT_SUM_OF_SQUARES (CATALOG #P24-0)

This unit is a generic function which computes the root sum of three squares; i.e., Result := Sqrt (X**2 + Y**2 + Z**2)

3.6.8.7.9.12.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R122.

3.6.8.7.9.12.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this unit:

| Name | Base Type | Description |
|---|---|---|
| Real_Type | generic float | Type of input and result variabls |
| Squared_Type | generic float | Type of intermediate result when a "Real_Type" is squared |

Subprograms:

The following table describes the generic formal subprograms required by this unit:

| Name | Type | Description |
|---|---|---|
| "*" | function | Squared_Type := Real_Type * Real_Type (used to perform a square function) |
| Sqrt | function | Real_Type := Square Root (Squared_Type) |

FORMAL PARAMETERS:

The following table describes this unit's formal parameters:

| Name | Type | Mode | Description |
|---|---|---|---|
| X | Real_Type | in | First of the three input vars |
| Y | Real_Type | in | Second of the three input vars |
| Z | Real_Type | in | Third of the three input vars |
| <return value> | Real_Type | out | Resultant root sum of squares |

3.6.8.7.9.12.3 INTERRUPTS

None.

3.6.8.7.9.12.4 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math, Basic_Data_Types;
procedure Sample is
   package BDT renames Basic_Data_Types;

   Result, X, Y, Z : BDT.Feet_Per_Second;

   function RSOS is new General_Purpose_Math.Root_Sum_Of_Squares
                        (Real_Type    => BDT.Feet_per_Second,
                         Squared_Type => BDT.Feet_Squared_Per_Second_Squared,
                         "*"          => BDT."*",
                         Sqrt         => BDT.Sqrt);
begin
   Result := RSOS (X => X,
                   Y => Y,
                   Z => Z);
end Sample;
```

3.6.8.7.9.12.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.7.9.12.6 DECOMPOSITION

None.

3.6.8.7.9.13 SIGN (CATALOG #P25-0)

This unit is a generic function which determines the sign of an input value; it returns -1 if input is negative, 1 if non-negative

3.6.8.7.9.13.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R224.

3.6.8.7.9.13.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this unit:

| Name | Base Type | Description |
|---|---|---|
| Real_Type | generic float | Type of input variable |

FORMAL PARAMETERS:

The following table describes this unit's formal parameters:

| Name | Type | Mode | Description |
|---|---|---|---|
| Input_Variable | Real_Type | in | Input to sign function |
| <returned value> | INTEGER | out | Integer representing the sign of Input_Variable |

3.6.8.7.9.13.3 INTERRUPTS

None.

3.6.8.7.9.13.4 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is
   type Real_Type    is digits 6;
   Result : INTEGER;
   Input  : Real_Type := 4.0;

   function Sgn is new General_Purpose_Math.Sign
                          (Real_Type => Real_Type);
begin
   Result := Sgn (Input => Input);
end Sample;
```

3.6.8.7.9.13.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.7.9.13.6 DECOMPOSITION

None.

3.6.8.7.9.14 MEAN_VALUE (CATALOG #P26-0)

This unit is a generic function which computes the average value of a vector of numbers.

3.6.8.7.9.14.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R144.

3.6.8.7.9.14.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this unit:

| Name | Base Type | Description |
|---|---|---|
| Element_Type | generic float | Type of the elements averaged |
| Index_Type | Discrete | Type of index to vector |
| Vector_Type | ARRAY | Array of "Element_Type" with "Index_Type" as the index |

FORMAL PARAMETERS:

The following table describes this unit's formal parameters:

| Name | Type | Mode | Description |
|---|---|---|---|
| Value_Vector | Vector_Type | in | Values to be averaged |
| <return value> | Element_Type | in | Average of input values |

3.6.8.7.9.14.3 INTERRUPTS

None.

3.6.8.7.9.14.4 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is
   type Elements is digits 6;
   type Index is range 1 .. 10;
   type Vector_Type is array (Index) of Elements;

   My_Vector : Vector_Type := (0, 5, 10, 15, 14, 13, 12, 11, 10, 20);
   Mean_Val  : Elements;

   function MV is new General_Purpose_Math.Mean_Value
                         (Element_Type => Elements,
                          Index_Type   => Index,
                          Vector_Type  => Vector_Type);
begin
```

```
      Mean_Val := MV (Value_Vector => My_Vector);
   end Sample;
```

3.6.8.7.9.14.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.7.9.14.6 DECOMPOSITION

None.

3.6.8.7.9.15 MEAN_ABSOLUTE_DIFFERENCE (CATALOG #P27-0)

This unit is a generic function which computes the mean absolute difference (MAD) of a vector, i.e., Avg (Abs (Xi - Xavg))

3.6.8.7.9.15.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R143.

3.6.8.7.9.15.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this unit:

| Name | Base Type | Description |
|---|---|---|
| Element_Type | generic float | Type of the elements averaged |
| Index_Type | Discrete | Type of index to vector |
| Vector_Type | ARRAY | Array of "Element_Type" with "Index_Type" as the index |

FORMAL PARAMETERS:

The following table describes this unit's formal parameters:

| Name | Type | Mode | Description |
|---|---|---|---|
| Value_Vector | Vector_Type | in | Input values |
| <return value> | Element_Type | in | MAD of input vector |

3.6.8.7.9.15.3 INTERRUPTS

None.

3.6.8.7.9.15.4 TIMING AND SEQUENCING

The following code illustrates a sample use of this part:

```
with General_Purpose_Math;
procedure Sample is
   type Elements is digits 6;
   type Index is range 1 .. 10;
   type Vector_Type is array (Index) of Elements;

   My_Vector : Vector_Type := (0, 5, 10, 15, 14, 13, 12, 11, 10, 20);
   Mean_Dif : Elements;

   function MAD is new General_Purpose_Math.Mean_Absolute_Difference
                          (Element_Type => Elements,
                           Index_Type   => Index,
                           Vector_Type  => Vector_Type);
begin
   Mean_Dif := MAD (Value_Vector => My_Vector);
end Sample;
```

3.6.8.7.9.15.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.7.9.15.6 DECOMPOSITION

None.

3.6.8.7.9.16 TWO_WAY_TABLE_LOOKUP (CATALOG #P1077-0)

This package provides a general two way table lookup. These routines allow the table to be created and initialized, or an already existing table may be used. Either variable type may be looked up in the table. The routines return a single value, interpolated or extrapolated as necessary.

3.6.8.7.9.16.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Two_Way_Table_Lookup | |

3.6.8.7.9.16.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Indices | INTEGER or Enumerated | Type to index the table |
| X_Values | FLOAT | Type of 1 table value |
| Y_Values | FLOAT | Type of other table value |
| Real | FLOAT | Type for intermediate calculations |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "/" | function | Divide operator for X_Values / Y_Values => Real |
| "/" | function | Divide operator for Y_Values / X_Values => Real |
| "/" | function | Divide operator for X_Values / X_Values => Y_Values |
| "/" | function | Divide operator for Y_Values / Y_Values => X_Values |
| "*" | function | Multiply operator for X_Values * Y_Values => Real |
| "*" | function | Multiply operator for Y_Values * X_Values => Real |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following chart describes the data types exported by this part:

| Name | Range | Description |
|---|---|---|
| X_Arrays | Indices | Array type for 1 type of values in table |
| Y_Arrays | Indices | Array type for other type of values in table |
| Tables | Indices | Type for Table of X and Y arrays |

Data objects:

The following chart describes the data objects exported by this part:

| Name | Type | Description |
|---|---|---|
| Table | record | Table of X_Array and Y_Array to be operated on |

3.6.8.7.9.16.3 LOCAL ENTITIES

None.

3.6.8.7.9.16.4 INTERRUPTS

None.

3.6.8.7.9.16.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Purpose_Math;
procedure sample is

   package GPM renames General_Purpose_Math;

   type Index_Range is range 1..50;
   type My_Xs is array( Index_Range ) of FLOAT;
   type My_YS is array( Index_Range ) of FLOAT;
   type My_Tables is
      record
          X : My_Xs;
          Y : My_Ys;
      end record;

   Degree_Value : FLOAT;
   Radian_Value : FLOAT;
   My_Table     : My_Tables

   Lookup is new GPM.Two_Way_Table_Lookup
                      ( Indices   => Index_Range,
                        X_Values  => FLOAT,
                        Y_Values  => FLOAT,
                        Real      => FLOAT );

   begin
      Lookup.Initialize( Table : My_Table,
                         Index : 25,
                         X     : 90.0,
                         Y     : 1.570796 );
      Radian_Value := Lookup.Lookup_Y( Table => My_Table,
                                       Input => 90.0 );
      Degree_Value := Lookup.Lookup_X( Table => My_Table,
                                       Input => 1.570796 );
   end Sample;
```

3.6.8.7.9.16.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.7.9.16.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Initialize | procedure | Allows the user to insert an X value and a Y value at a particular index in the table |
| Lookup_Y | function | Given an X value, this function finds the corresponding Y value, interpolating or extrapolating as necessary |
| Lookup_X | function | Given a Y value, this function find the corresponding X value, interpolating or extrapolating as necessary |

3.6.8.7.9.16.8 PART DESIGN

None.

```
package General_Purpose_Math is

pragma PAGE;
   generic
      type Independent_Type is digits <>;
      type Dependent_Type   is digits <>;
      type Index_Type       is (<>);
      Minimum_Independent_Value : Independent_Type;
      Maximum_Independent_Value : Independent_Type;
   package Lookup_Table_Even_Spacing is

      Value_Out_Of_Range : exception;

      type Key_Range_Flag is (Below_Table_Range, In_Table_Range, Above_Table_Range);

      type Tables is array (Index_Type) of Dependent_Type;

      procedure Initialize (Table           : out Tables;
                            INDEX           : in  Index_Type;
                            Dependent_Value : in  Dependent_Type);

      procedure Lookup (Table             : in  Tables;
                        Key               : in  Independent_Type;
                        Lower_Independent : out Independent_Type;
                        Higher_Independent: out Independent_Type;
                        Lower_Dependent   : out Dependent_Type;
                        Higher_Dependent  : out Dependent_Type);

      procedure Lookup (Table             : in  Tables;
                        Key               : in  Independent_Type;
                        Lower_Independent : out Independent_Type;
                        Higher_Independent: out Independent_Type;
                        Lower_Dependent   : out Dependent_Type;
                        Higher_Dependent  : out Dependent_Type;
                        Key_Location      : out Key_Range_Flag);

   end Lookup_Table_Even_Spacing;

pragma PAGE;
   generic
      type Independent_Type is digits <>;
      type Dependent_Type   is digits <>;
      type Index_Type       is (<>);
   package Lookup_Table_Uneven_Spacing is

      Value_Out_Of_Range : exception;

      type Key_Range_Flag is (Below_Table_Range, In_Table_Range, Above_Table_Range);

      type Table_Entries is
         record
            Independent_Entry : Independent_Type;
            Dependent_Entry   : Dependent_Type;
         end record;

      type Tables is array (Index_Type) of Table_Entries;
```

```
      procedure Initialize (Table             : out Tables;
                            INDEX             : in  Index_Type;
                            Independent_Value : in  Independent_Type;
                            Dependent_Value   : in  Dependent_Type);

      procedure Lookup (Table             : in  Tables;
                        Key               : in  Independent_Type;
                        Lower_Independent : out Independent_Type;
                        Higher_Independent: out Independent_Type;
                        Lower_Dependent   : out Dependent_Type;
                        Higher_Dependent  : out Dependent_Type);

      procedure Lookup (Table             : in  Tables;
                        Key               : in  Independent_Type;
                        Lower_Independent : out Independent_Type;
                        Higher_Independent: out Independent_Type;
                        Lower_Dependent   : out Dependent_Type;
                        Higher_Dependent  : out Dependent_Type;
                        Key_Location      : out Key_Range_Flag);

   end Lookup_Table_Uneven_Spacing;

pragma PAGE;
   generic
      type Real_Type is digits <>;
      Initial_Value    : in Real_Type := 0.0;
      Increment_Amount : in Real_Type := 1.0;
   package Incrementor is

      procedure Reinitialize (Initial_Value    : in Real_Type;
                              Increment_Amount : in Real_Type);

      function Increment return Real_Type;

   end Incrementor;

pragma PAGE;
   generic
      type Real_Type is digits <>;
      Initial_Value    : in Real_Type := 0.0;
      Decrement_Amount : in Real_Type := 1.0;
   package Decrementor is

      procedure Reinitialize (Initial_Value    : in Real_Type;
                              Decrement_Amount : in Real_Type);

      function Decrement return Real_Type;

   end Decrementor;

pragma PAGE;
   generic
      type Real_Type is digits <>;
      Initial_Sum   : in Real_Type := 0.0;
      Initial_Count : in INTEGER   := 0;
      with function "/" (Left : Real_Type; Right : INTEGER)
                        return Real_Type is <>;
```

```
   package Running_Average is

      procedure Reinitialize (Initial_Sum   : in Real_Type;
                              Initial_Count : in INTEGER);

      procedure Reinitialize (Initial_Count : in INTEGER);

      function Current_Average (New_Value : Real_Type) return Real_Type;

   end Running_Average;

pragma PAGE;
   generic
      type Element_Type is digits <>;
      Initial_Value : in Element_Type := 0.0;
   package Accumulator is

      procedure Reinitialize (Initial_Value : in Element_Type);

      procedure Accumulate (New_Value : in Element_Type);

      procedure Accumulate (New_Value       : in  Element_Type;
                            Retrieved_Value : out Element_Type);

      function Retrieve return Element_Type;

   end Accumulator;

pragma PAGE;
   generic
      type Element_Type is digits <>;
      Initial_Value : in Element_Type := 0.0;
   package Change_Calculator is

      procedure Reinitialize (Initial_Value : in Element_Type);

      function Change (New_Value : Element_Type) return Element_Type;

      function Retrieve_Value return Element_Type;

   end Change_Calculator;

pragma PAGE;
   generic
      type Element_Type is digits <>;
      Initial_Accumulator_Value : in Element_Type := 0.0;
      Initial_Previous_Value    : in Element_Type := 0.0;
   package Change_Accumulator is

      procedure Reinitialize (Initial_Accumulator_Value : in Element_Type);

      procedure Reinitialize (Initial_Accumulator_Value : in Element_Type;
                              Initial_Previous_Value    : in Element_Type);

      procedure Accumulate_Change (New_Value : in Element_Type);

      procedure Accumulate_Change (New_Value                    : in  Element_Type;
```

```
                                    Retrieved_Accumulator_Value : out Element_Type);

      function Retrieve_Accumulation return Element_Type;

      function Retrieve_Previous_Value return Element_Type;

   end Change_Accumulator;

pragma PAGE;
   generic
      type Dependent_Type   is digits <>;
      type Independent_Type is digits <>;
      type Time_Interval    is digits <>;
      Initial_Dependent_Value   : in Dependent_Type;
      Initial_Independent_Value : in Independent_Type;
      Default_Delta_Time        : in Time_Interval;
      with function "*" (Left : Independent_Type; Right : Time_Interval)
                        return Dependent_Type is <>;
   package Integrator is

      procedure Reinitialize (Initial_Dependent_Value   : in Dependent_Type;
                              Initial_Independent_Value : in Independent_Type);

      procedure Update (Current_Independent_Value : in Independent_Type);

      function Integrate (Current_Independent_Value : Independent_Type;
                          Delta_Time                : Time_Interval
                                                      := Default_Delta_Time)
                        return Dependent_Type;

   end Integrator;

pragma PAGE;
   generic
      type Independent_Type                is digits <>;
      type Dependent_Type                  is digits <>;
      with function "/" (Left  : Independent_Type;
                         Right : Independent_Type)
                        return Dependent_Type is <>;
   function Interpolate_Or_Extrapolate
               (Input             : in Independent_Type;
                Lower_Independent : in Independent_Type;
                Higher_Independent : in Independent_Type;
                Lower_Dependent   : in Dependent_Type;
                Higher_Dependent  : in Dependent_Type)
               return Dependent_Type;

pragma PAGE;
   generic
      type Inputs  is digits <>;
      type Outputs is digits <>;
      type Real    is digits <>;
   package Square_Root is

      Negative_Input : exception;

      function Sqrt (Input : Inputs) return Outputs;
```

```
   end Square_Root;

pragma PAGE;
   generic
      type Real_Type is digits <>;
      type Squared_Type is digits <>;
      with function "*" (Left : Real_Type; Right : Real_Type)
                        return Squared_Type is <>;
      with function Sqrt (Input : Squared_Type) return Real_Type is <>;
   function Root_Sum_Of_Squares (X : Real_Type;
                                 Y : Real_Type;
                                 Z : Real_Type)
                                return Real_Type;

pragma PAGE;
   generic
      type Real_Type is digits <>;
   function Sign (Input_Variable : Real_Type)
                 return INTEGER;

pragma PAGE;
   generic
      type Element_Type is digits <>;
      type Index_Type   is (<>);
      type Vector_Type  is array (Index_Type range <>) of Element_Type;
   function Mean_Value (Value_Vector : Vector_Type) return Element_Type;

pragma PAGE;
   generic
      type Element_Type is digits <>;
      type Index_Type   is (<>);
      type Vector_Type  is array (Index_Type range <>) of Element_Type;
   function Mean_Absolute_Difference (Value_Vector : Vector_Type)
                                     return Element_Type;

pragma PAGE;
   generic
      type Indices  is (<>);
      type X_Values is digits <>;
      type Y_Values is digits <>;
      type Real     is digits <>;
      with function "/" (Left  : X_Values;
                         Right : Y_Values) return Real     is <>;
      with function "/" (Left  : Y_Values;
                         Right : X_Values) return Real     is <>;
      with function "/" (Left  : X_Values;
                         Right : X_Values) return Y_Values is <>;
      with function "/" (Left  : Y_Values;
                         Right : Y_Values) return X_Values is <>;
      with function "*" (Left  : X_Values;
                         Right : Y_Values) return Real     is <>;
      with function "*" (Left  : Y_Values;
                         Right : X_Values) return Real     is <>;

   package Two_Way_Table_Lookup is
```

```
      type X_Arrays is array( Indices ) of X_Values;
      type Y_Arrays is array( Indices ) of Y_Values;

      type Tables is
         record
            Table_X : X_Arrays;
            Table_Y : Y_Arrays;
         end record;

      Table : Tables;

      procedure Initialize( Table : out Tables;
                            INDEX : in  Indices;
                            X     : in  X_Values;
                            Y     : in  Y_Values);

      function Lookup_Y ( Table : Tables;
                          Input : X_Values ) return Y_Values;

      function Lookup_X ( Table : Tables;
                          Input : Y_Values ) return X_Values;

   end Two_Way_Table_Lookup;


end General_Purpose_Math;
```

3.6.8.8 POLYNOMIALS TLCSC (CATALOG #P885-0)

This part is a package of packages. It contains specifications for all the polynomial functions required by the rest of the CAMP parts.

Each subpackage, except General_Polynomial, contains function(s) for one type of polynomial (i.e. Hastings, Taylor series, etc.). There is also a package, System_Functions, which provides access to the Ada system run-time math library.

These parts provide standard mathematical functions such as trigonometric and square root functions. For these parts, the term "standard mathematical functions" refers to:

- o Sine (x)
- o Cosine (x)
- o Tangent (x)
- o Arcsine (x)
- o Arccosine (x)
- o Arctangent (x)
- o Square root (x)
- o Log base 10 (x)
- o Log base n (x)

These functions can be accessed in one of the following ways:

- o A standard set of polynomial solutions can be obtained by with'ing the Basic Data types part (P621) or instantiation of the Trigonometric part (P683).
- o Selected polynomial solutions may be obtained by with'ing the Polynomial part and instantiating the desired functions or packages.

In addition, a General_Polynomial package is provided which allows the creation of a user-defined polynomial function.

Many of these packages have functions which are identical except for the number of terms used in the calculations. These functions run from 4 term calculations to 8 term calculations. These functions can be instantiated with parameters which are either single or extended precision. However, the algorithms are such that only a certain amount of precision can be generated regardless of single or extended precision usage. The following general rules should be applied when determining whether to instantiate with single or extended precision.

- o Single precision functions should NOT be instantiated with MORE than 5 terms. Using single precision, no more precision is generated using more than 5 terms.
- o Extended precision functions should NOT be instantiated with LESS than 5 terms. More precision is not gained by using extended precision instead of single precision with less than 5 terms.
- o Given the above restrictions, the more terms a function has, the greater the precision of the result.

Exceptions to this rule are the square root functions, which have from 2 to 5 terms. They may be used either single or double precision without restrictions, although with single precision the higher term functions will generate more precision than the single precision variable can hold.

3.6.8.8.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Chebyshev | R214 |
| Cody_Waite | |
| Continued_Fractions | |
| Fike | R215 |
| General_Polynomial | partially meets CAMP requirements R214 thru R222 |
| Hart | R216 |
| Hastings | R217 |
| Modified_ Newton_Raphson | R220 |
| Newton_Raphson | R221 |
| System_Functions | R223 |
| Taylor_Series | R222 |

3.6.8.8.2 INPUT/OUTPUT

None.

3.6.8.8.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.8.8.4 LOCAL ENTITIES

None.

3.6.8.8.5 INTERRUPTS

None.

3.6.8.8.6 TIMING AND SEQUENCING

None.

3.6.8.8.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.8.8.8 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Chebyshev | package | Contains generic functions providing Chebyshev polynomial solutions to a set of standard mathematical functions |
| Continued_ Fractions | package | Contains generic functions providing Continued Fractions polynomial solutions for the tangent and arctangent functions |
| Cody_Waite | package | Contains generic functions providing Cody_Waite polynomial solutions to a set of standard mathematical functions |
| Fike | package | Contains generic functions providing Fike polynomial solutions to a set of standard mathematical functions |
| Hart | package | Contains generic functions providing Hart polynomial solutions to a set of standard mathematical functions |
| Hastings | package | Contains generic functions providing Hastings polynomial solutions to a set of standard mathematical functions |
| Modified_ Newton_Raphson | package | Contains generic functions providing Modified Newton-Raphson polynomial solutions to a set of standard mathematical functions |
| Newton_Raphson | package | Contains generic functions providing Newton-Raphson polynomial solutions to a set set of standard mathematical functions |
| Taylor_Series | package | Contains generic functions providing Taylor-Series polynomial solutions to a set of standard mathematical functions |
| General_ Polynomial | generic package | Allows the user to define a polynomial function and then to solve the user-polynomial for a given input value |
| System_Functions | package | Provides access to the Ada system library for standard mathematical functions |

3.6.8.8.9 PART DESIGN

3.6.8.8.9.1 CHEBYSHEV (CATALOG #P886-0)

This package contains a generic function providing a Chebyshev polynomial solution for the sine function. Provisions are made for the trigonometric functions to handle units of radians, semicircles, or degrees, respectively. Outputs are of type sin_cos_ratio.

3.6.8.8.9.1.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Chebyshev | R214 |

3.6.8.8.9.1.2 INPUT/OUTPUT

None.

3.6.8.8.9.1.3 LOCAL ENTITIES

None.

3.6.8.8.9.1.4 INTERRUPTS

None.

3.6.8.8.9.1.5 TIMING AND SEQUENCING

None.

3.6.8.8.9.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Chebyshev_ Radian_ Operations | generic package | Sine functions dealing with input in units of radians |
| Chebyshev_ Degree_ Operations | generic package | Sine functions dealing with input in units of degrees |
| Chebyshev_ Semicircle_ Operations | generic package | Sine functions dealing with input in units of semicircles |

3.6.8.8.9.1.8 PART DESIGN

3.6.8.8.9.1.8.1 CHEBYSHEV_RADIAN_OPERATIONS (CATALOG #P887-0)

This package contains a generic function providing a Chebyshev polynomial solution for the sine function. This package is designed to accept inputs in terms of radians.

3.6.8.8.9.1.8.1.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Chebyshev_Radian_Operations | This package partially fulfills R214 |

3.6.8.8.9.1.8.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Radians | Floating point | Allows floating point representation of radian measurements. |
| Real | Floating point | General floating point representation. |
| Sin_Cos_Ratio | Floating point | Represents sines and cosines. |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| One_Over_Pi | Radians | constant | constant value of inverse of Pi |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply radians * radians yielding a real result. |

3.6.8.8.9.1.8.1.3 LOCAL ENTITIES

None.

3.6.8.8.9.1.8.1.4 INTERRUPTS

None.

3.6.8.8.9.1.8.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;

procedure Sample is
   type Angles   is digits 6;
   type FPs      is digits 6;
   type Sines    is digits 6;

   One_Over_Pi : constant := 0.318310;
   Pi          : constant := 3.14159;

   Right_Angle : Angle;
   Sine_Result : Sines;

   function "*" ( Left_Side  : Angle;
                  Right_Side : Angle) return FPs;

   package Chebyshev_Sine is new Polynomials.Chebyshev_Radian_Operations
                                          ( Radians       => Angles,
                                            Real          => FPs,
                                            Sin_Cos_Ratio => Sines,
                                            One_Over_Pi   => One_Over_Pi,
                                            *             => * );
   begin
      Right_Angle := Pi / 2.0;
      Sine_Result := Chebyshev_Sine.Sin_R_5term( Right_Angle );
   end Sample;
```

3.6.8.8.9.1.8.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.1.8.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Sin_R_5term | function | Returns the sine of an angle computed with 5 terms, single or extended precision. |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Type | Catalog # |
|---|---|---|
| Sin_R_5term | function | P888-0 |

3.6.8.8.9.1.8.1.8 PART DESIGN

None.

3.6.8.8.9.1.8.2 CHEBYSHEV_DEGREE_OPERATIONS (CATALOG #P889-0)

This package contains a generic function providing a Chebyshev polynomial solution for the sine function. This package is designed to accept inputs in terms of degrees.

3.6.8.8.9.1.8.2.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Chebyshev_Degree_Operations | This package partially fulfills R214 |

3.6.8.8.9.1.8.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Degrees | Floating point | Allows floating point representation of degree measurements. |
| Real | Floating point | General floating point representation. |
| Sin_Cos_Ratio | Floating point | Represents sines and cosines. |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply degrees * degrees yielding a real result. |

3.6.8.8.9.1.8.2.3 LOCAL ENTITIES

None.

3.6.8.8.9.1.8.2.4 INTERRUPTS

None.

3.6.8.8.9.1.8.2.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;

procedure Sample is
   type Angles   is digits 6;
   type FPs      is digits 6;
   type Sines    is digits 6;

   Right_Angle : Angle;
   Sine_Result : Sines;

   function "*" ( Left_Side  : Angle;
                  Right_Side : Angle) return FPs;

   package Cheby_Deg is new Polynomials.Chebyshev_Degree_Operations
                                      ( Degrees       => Angles,
                                        Real          => FPs,
                                        Sin_Cos_Ratio => Sines,
                                        *             => * );
   begin
      Right_Angle := 90.0;
      Sine_Result := Cheby_Deg.Sin_R_5term( Right_Angle );
```

```
end Sample;
```

3.6.8.8.9.1.8.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.1.8.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Sin_R_5term | function | Returns the sine of an angle computed with 5 terms, single or extended precision. |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Type | Catalog # |
|---|---|---|
| Sin_R_5term | function | P890-0 |

3.6.8.8.9.1.8.2.8 PART DESIGN

None.

3.6.8.8.9.1.8.3 CHEBYSHEV_SEMICIRCLE_OPERATIONS (CATALOG #P891-0)

This package contains a generic function providing a Chebyshev polynomial solution for the sine function. This package is designed to accept inputs in terms of semicircles.

3.6.8.8.9.1.8.3.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Chebyshev_Semicircle_Operations | This package partially fulfills R214 |

3.6.8.8.9.1.8.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Semicircles | Floating point | Allows floating point representation of semicircle measurements. |
| Real | Floating point | General floating point representation. |
| Sin_Cos_Ratio | Floating point | Represents sines and cosines. |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply semicircles * semicircles yielding a real result. |

3.6.8.8.9.1.8.3.3 LOCAL ENTITIES

None.

3.6.8.8.9.1.8.3.4 INTERRUPTS

None.

3.6.8.8.9.1.8.3.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;

procedure Sample is
   type Angles   is digits 6;
   type FPs      is digits 6;
   type Sines    is digits 6;

   Right_Angle : Angle;
   Sine_Result : Sines;

   function "*" ( Left_Side  : Angle;
                  Right_Side : Angle) return FPs;
```

```
package Cheby_Semi is new Polynomials.Chebyshev_Semicircle_Operations
                                        ( Semicircles   => Angles,
                                          Real          => FPs,
                                          Sin_Cos_Ratio => Sines,
                                          *             => * );
begin
   Right_Angle := 0.5;
   Sine_Result := Cheby_Semi.Sin_R_5term( Right_Angle );
end Sample;
```

3.6.8.8.9.1.8.3.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.1.8.3.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Sin_R_5term | function | Returns the sine of an angle computed with 5 terms, single or extended precision. |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Type | Catalog # |
|---|---|---|
| Sin_R_5term | function | P892-0 |

3.6.8.8.9.1.8.3.8 PART DESIGN

None.

3.6.8.8.9.2 CODY_WAITE (CATALOG #P893-0)

This package contains a generic function providing a Cody_Waite polynomial solution for the log functions.

3.6.8.8.9.2.1 REQUIREMENTS ALLOCATION

None.

3.6.8.8.9.2.2 INPUT/OUTPUT

None.

3.6.8.8.9.2.3 LOCAL ENTITIES

None.

3.6.8.8.9.2.4 INTERRUPTS

None.

3.6.8.8.9.2.5 TIMING AND SEQUENCING

None.

3.6.8.8.9.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Cody_Natural_Log | generic package | Natural log functions. |
| Cody_Log_Base_N | generic package | Log functions to base N. |

3.6.8.8.9.2.8 PART DESIGN

3.6.8.8.9.2.8.1 CODY_NATURAL_LOG (CATALOG #P894-0)

This generic package contains functions providing Cody Waite polynomial solutions for the natural log function.

3.6.8.8.9.2.8.1.1 REQUIREMENTS ALLOCATION

None.

3.6.8.8.9.2.8.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Inputs | Floating point | Floating point input to the function |
| Outputs | Floating point | Floating point output to the function |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply Inputs * Inputs yielding a result of type Outputs. |

3.6.8.8.9.2.8.1.3 LOCAL ENTITIES

None.

3.6.8.8.9.2.8.1.4 INTERRUPTS

None.

3.6.8.8.9.2.8.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;

procedure Sample is
   type FPs      is digits 9;

   Sample_Num : FPs;
   Result     : FPs;

   package Nat_Log is new Polynomials.Cody_Natural_Log
                                          ( Inputs  => FPs;
                                            Outputs => FPs);
   begin
      Sample_Num := 33.0
      Result := Nat_Log.Nat_Log( Sample_Num );
   end Sample;
```

3.6.8.8.9.2.8.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.2.8.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Defloat | procedure | Reduces the real number x to sign * mantissa * 2 ** exponent |
| Nat_Log | function | Returns the natural logarithm of a number |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Type | Catalog # |
|---|---|---|
| Nat_Log | function | P895-0 |

3.6.8.8.9.2.8.1.8 PART DESIGN

None.

3.6.8.8.9.2.8.2 CODY_LOG_BASE_N (CATALOG #P896-0)

This generic package contains functions providing Cody Waite polynomial solutions for the log function for base N.

3.6.8.8.9.2.8.2.1 REQUIREMENTS ALLOCATION

None

3.6.8.8.9.2.8.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Inputs | Floating point | Floating point input to the function |
| Outputs | Floating point | Floating point output to the function |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Base_N | Positive | defualt = 10 | Base to operate in |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply Inputs * Inputs yielding a result of type Outputs. |

3.6.8.8.9.2.8.2.3 LOCAL ENTITIES

None.

3.6.8.8.9.2.8.2.4 INTERRUPTS

None.

3.6.8.8.9.2.8.2.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;

procedure Sample is
   type FPs      is digits 9;

   Base       : Positive;
   Sample_Num : FPs;
   Result     : FPs;

   package Log_Base_5 is new Polynomials.Cody_Log_Base_N
                                          ( Inputs  => FPs;
                                            Outputs => FPs,
                                            Base_N  => Base );
   begin
      Base := 5;
      Sample_Num := 35.0
      Result := Cody_Log_Base_N.Log_N ( Sample_Num );
   end Sample;
```

3.6.8.8.9.2.8.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.2.8.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Log_Base_N | function | Returns the logarithm to Base of a number computed with 8 terms, either precision |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Type | Catalog # |
|---|---|---|
| Log_Base_N | function | P897-0 |

3.6.8.8.9.2.8.2.8 PART DESIGN

None.

3.6.8.8.9.3 CONTINUED_FRACTIONS (CATALOG #P898-0)

This package contains generic functions providing Continued Fractions polynomial solutions for the Tangent and Arctangent functions. Provisions are made for the trigonometric functions to handle units of radians.

3.6.8.8.9.3.1 REQUIREMENTS ALLOCATION

None.

3.6.8.8.9.3.2 INPUT/OUTPUT

None.

3.6.8.8.9.3.3 LOCAL ENTITIES

None.

3.6.8.8.9.3.4 INTERRUPTS

None.

3.6.8.8.9.3.5 TIMING AND SEQUENCING

None.

3.6.8.8.9.3.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.3.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Continued_ Radian_ Operations | generic package | Tangent and arctangent functions dealing with input in terms of radians |

3.6.8.8.9.3.8 PART DESIGN

3.6.8.8.9.3.8.1 CONTINUED_RADIAN_OPERATIONS (CATALOG #P899-0)

This generic packages contains functions providing Continued Fractions polynomial solutions for the tangent and arctangent functions. This package is designed to handle units of radians.

3.6.8.8.9.3.8.1.1 REQUIREMENTS ALLOCATION

None.

3.6.8.8.9.3.8.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Radians | Floating Point | Angle expressed radians |
| Tan_Ratio | Floating Point | Value of computed tangent function |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Default_Term_Count | Positive | Number of terms in the calculation |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply radians * radians yielding a tan_ratio result. |

### 3.6.8.8.9.3.8.1.3 LOCAL ENTITIES

None.

### 3.6.8.8.9.3.8.1.4 INTERRUPTS

None.

### 3.6.8.8.9.3.8.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;

procedure Sample is
   type Angles   is digits 6;
   type Tangents is digits 6;

   Count : Positive;
   Right_Angle : Angle;
   Result : Tangents;

   function "*" ( Left_Side  : Angle;
                  Right_Side : Angle) return Tangents;

   package Continued_Rad is new Polynomials.Continued_Radian_Operations
                                      ( Radians           => Angles,
                                        Tan_Ratio         => Tangents,
                                        Default_Term_Count => Count,
                                        *                 => * );
   begin
      Right_Angle := Pi / 2.0;
```

```
      Result := Continued_Rad.Tan_R_5term( Right_Angle );
   end Sample;
```

3.6.8.8.9.3.8.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.3.8.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Tan_R | generic function | Tangent function dealing with input in units of radians |
| Arctan_R | generic function | Arctangent function dealing with output in units of radians |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Type | Catalog # |
|---|---|---|
| Tan_R | generic function | P900-0 |
| Arctan_R | generic function | P901-0 |

3.6.8.8.9.3.8.1.8 PART DESIGN

None.

3.6.8.8.9.4 FIKE (CATALOG #P902-0)

This packages contains generic functions providing Fike polynomial solutions for the arcsine function. Provisions are made for the arcsine functions to accept units of sin_cos_ratio. Outputs are of type semicircles.

3.6.8.8.9.4.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Fike | R215 |

3.6.8.8.9.4.2 INPUT/OUTPUT

None.

3.6.8.8.9.4.3 LOCAL ENTITIES

None.

3.6.8.8.9.4.4 INTERRUPTS

None.

3.6.8.8.9.4.5 TIMING AND SEQUENCING

None.

3.6.8.8.9.4.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.4.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Fike_Semicircle_Operations | generic package | Arcine functions dealing with input in units of semicircles |

3.6.8.8.9.4.8 PART DESIGN

3.6.8.8.9.4.8.1 FIKE_SEMICIRCLE_OPERATIONS (CATALOG #P903-0)

This generic package contains a function providing Fike a polynomial solution for the arcsine function. This package is designed to accept inputs in terms of sin_cos_ratio.

3.6.8.8.9.4.8.1.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Fike_Semicircle_Operations | This package partially fulfills R215 |

3.6.8.8.9.4.8.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Semicircles | Floating point | Allows floating point representation of semicircle measurements. |
| Real | Floating point | General floating point representation. |
| Sin_Cos_Ratio | Floating point | Represents sines and cosines. |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| Sqrt | funtion | returns the square root of type real |

3.6.8.8.9.4.8.1.3 LOCAL ENTITIES

None.

3.6.8.8.9.4.8.1.4 INTERRUPTS

None.

3.6.8.8.9.4.8.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;

procedure Sample is
   type Angles is digits 6;
   type FPs    is digits 6;
   type Sines  is digits 6;

   Right_Angle : Angle;
   Sine_Result : Sines;

   function Sqrt ( Input : Real) return Real;

   package Fike_Semi is new Polynomials.Fike_Semicircle_Operations
                                    ( Semicircles   => Angles,
```

```
                                        Sin_Cos_Ratio => Sines,
                                        Real          => FPs,
                                        Sqrt          => Sqrt );
   begin
      Right_Angle := 1.0;
      Sine_Result := Fike_Semi.Arcsin_R_6term( Right_Angle );
   end Sample;
```

3.6.8.8.9.4.8.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.4.8.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Arcsin_S_6term | function | Returns the sine of an angle computed with 6 terms single or extended precision. |
| Arccos_S_6term | function | Returns the cosine of an angle computed with 6 terms, single or extended precision. |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Type | Catalog # |
|---|---|---|
| Arcsin_S_6term | function | P904-0 |
| Arccos_S_6term | function | P905-0 |

3.6.8.8.9.4.8.1.8 PART DESIGN

None.

3.6.8.8.9.5 GENERAL_POLYNOMIAL (CATALOG #P906-0)

This package allows the user to define a polynomial function and to then solve the user-polynomial for a given input value.

3.6.8.8.9.5.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| General_Polynomial | partially meets R214 through R222 |

3.6.8.8.9.5.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Inputs | floating point type | Data type of independent values |
| Results | floating point type | Data type of dependent values |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Coefficient_Count | Positive | Number of coefficient in the polynomial |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "**" | function | Exponential operator defining the operation:<br>Inputs ** x := Results |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Data types:

The following chart describes the data types exported by this part:

| Name | Range | Operators | Description |
|---|---|---|---|
| Coefficient_ Records | N/A | N/A | Contains the a and b components of a polynomial term:a*(x**b) |
| Table_ Dimensions | 1 .. Coefficient_ Count | N/A<br>N/A | Defines the size of the polynomial table |

Data objects:

The following table describes the data objects exported by this part:

| Name | Type | Definition |
|---|---|---|
| Polynomial_ Definition | array | Array of polynomial terms |

3.6.8.8.9.5.3 LOCAL ENTITIES

None.

3.6.8.8.9.5.4 INTERRUPTS

None.

3.6.8.8.9.5.5 TIMING AND SEQUENCING

The following is a sample usage of this part:

```
with Polynomials;
...
   type Dependent_Values   is new FLOAT;
   type Independent_Values is new FLOAT;
   ...
   function "**" (Left  : Independent_Values,
                  Right : POSITIVE) return FLOAT;
   ...
   package Compute_New_Value is new
           General_Polynomial (Inputs           => Independent_Values,
                               Results          => Dependent_Values,
                               Coefficient_Count => 3);
   ...
   function My_Compute renames Compute_New_Value.Polynomial;
   ...
   a : constant FLOAT    := 1.5;
   b : constant FLOAT    := 2.5;
   c : constant FLOAT    := 3.5;
   d : constant POSITIVE := 2;
   ...
   begin
```

```
...
--create table to calculate the polynomial:
--f(x) := a + b*x + c*(x**d)

Compute_New_Value.
   Polynomial_Definition(1) := (Coefficient => a,
                                Power_of_X  => 0);
Compute_New_Value.
   Polynomial_Definition(2) := (Coefficient => b,
                                Power_of_X  => 1);
Compute_New_Value.
   Polynomial_Definition(3) := (Coefficient => c,
                                Power_of_X  => d);
...
Result := My_Compute( X );
```

3.6.8.8.9.5.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.5.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Polynomial | function | Calculates f(x) where f is defined by the polynomial_definition table |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Type | Catalog # |
|---|---|---|
| Polynomial | function | P907-0 |

3.6.8.8.9.5.8 PART DESIGN

None.

3.6.8.8.9.6 HART (CATALOG #P908-0)

This packages contains generic functions providing Hart polynomial solutions for the cosine function. Provisions are made for the trigonometric functions to handle units of radians or degrees, respectively. Outputs may be of type sin_cos_ratio.

3.6.8.8.9.6.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Hart | R216 |

3.6.8.8.9.6.2 INPUT/OUTPUT

None.

3.6.8.8.9.6.3 LOCAL ENTITIES

None.

3.6.8.8.9.6.4 INTERRUPTS

None.

3.6.8.8.9.6.5 TIMING AND SEQUENCING

None.

3.6.8.8.9.6.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.6.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Hart_Radian_Operations | generic package | Cosine functions dealing with input in units of radians |
| Hart_Degree_Operations | generic package | Sine functions dealing with input in units of degrees |

3.6.8.8.9.6.8 PART DESIGN

3.6.8.8.9.6.8.1 HART_RADIAN_OPERATIONS (CATALOG #P909-0)

This generic package contains a function providing a Hart polynomial solution for the cosine function. This package is designed to accept inputs in terms of radians.

3.6.8.8.9.6.8.1.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Hart_Radian_Operations | This package partially fulfills R216 |

3.6.8.8.9.6.8.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Radians | Floating point | Allows floating point representation of radian measurements. |
| Real | Floating point | General floating point representation. |
| Sin_Cos_Ratio | Floating point | Represents sines and cosines. |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply radians * radians yielding a real result. |

3.6.8.8.9.6.8.1.3 LOCAL ENTITIES

None.

3.6.8.8.9.6.8.1.4 INTERRUPTS

None.

3.6.8.8.9.6.8.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
   with Polynomials;

   procedure Sample is
      type Angles   is digits 6;
      type FPs      is digits 6;
      type Sines    is digits 6;
      type Tangents is digits 6;

      Pi          : constant := 3.14159;
      One_Over_Pi : constant := 1.0/Pi;

      Right_Angle : Angle;
      Cosine_Result : Sines;

      function "*" ( Left_Side  : Angle;
                     Right_Side : Angle) return FPs;

      package Hart_Radian is new Polynomials.Hart_Radian_Operations
                                        ( Radians       => Angles,
                                          Real          => FPs,
                                          Sin_Cos_Ratio => Sines,
                                          Pi            => Pi,
                                          One_Over_Pi   => One_Over_Pi,
                                          *             => * );
   begin
      Right_Angle := Pi / 2.0;
      Cosine_Result := Hart_Cosine.Cos_R_5term( Right_Angle );
   end Sample;
```

3.6.8.8.9.6.8.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.6.8.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Cos_R_5term | function | Returns the cosine of an angle computed with 5 terms, single or extended precision. |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Type | Catalog # |
|---|---|---|
| Cos_R_5term | function | P910-0 |

3.6.8.8.9.6.8.1.8 PART DESIGN

None.

3.6.8.8.9.6.8.2 HART_DEGREE_OPERATIONS (CATALOG #P911-0)

This generic package contains a function providing a Hart polynomial solution for the cosine function. This package is designed to accept inputs in terms of degrees.

3.6.8.8.9.6.8.2.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Hart_Degree_Operations | This package partially fulfills R216 |

3.6.8.8.9.6.8.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Degrees | Floating point | Allows floating point representation of degree measurements. |
| Real | Floating point | General floating point representation. |
| Sin_Cos_Ratio | Floating point | Represents sines and cosines. |
| Tan_Ratio | Floating point | Represents tangent values. |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply degrees * degrees yielding a real result. |

3.6.8.8.9.6.8.2.3 LOCAL ENTITIES

None.

3.6.8.8.9.6.8.2.4 INTERRUPTS

None.

3.6.8.8.9.6.8.2.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;

procedure Sample is
   type Angles   is digits 6;
   type FPs      is digits 6;
   type Sines    is digits 6;
   type Tangents is digits 6;

   Pi           : constant := 3.14159;

   Right_Angle : Angle;
   Cosine_Result : Sines;

   function "*" ( Left_Side  : Angle;
                  Right_Side : Angle) return FPs;

   package Hart_Degree is new Polynomials.Hart_Degree_Operations
                                    ( Degrees       => Angles,
                                      Real          => FPs,
                                      Sin_Cos_Ratio => Sines,
                                      *             => * );
begin
   Right_Angle := 90.0;
   Cosine_Result := Hart_Degree.Cos_R_5term( Right_Angle );
end Sample;
```

3.6.8.8.9.6.8.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.6.8.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Cos_R_5term | function | Returns the cosine of an angle computed with 5 terms, single or extended precision. |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Type | Catalog # |
|---|---|---|
| Cos_R_5term | function | P912-0 |

3.6.8.8.9.6.8.2.8 PART DESIGN

None.

3.6.8.8.9.7 HASTINGS (CATALOG #P913-0)

This package contains generic functions providing Hastings polynomial solutions for a set of trigonometric functions, which include sine, cosine, tangent, and arctangent. Provisions are made for the trigonometric functions to handle units of radians or degrees.

3.6.8.8.9.7.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Hastings | R217 |

3.6.8.8.9.7.2 INPUT/OUTPUT

None.

3.6.8.8.9.7.3 LOCAL ENTITIES

None.

3.6.8.8.9.7.4 INTERRUPTS

None.

3.6.8.8.9.7.5 TIMING AND SEQUENCING

None.

3.6.8.8.9.7.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.7.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Hastings_Radian_Operations | generic package | Trigonometric functions dealing with input in units of radians |
| Hastings_Degree_Operations | generic package | Trigonometric functions dealing with input in units of degrees |

3.6.8.8.9.7.8 PART DESIGN

3.6.8.8.9.7.8.1 HASTINGS_RADIAN_OPERATIONS (CATALOG #P914-0)

This generic package contains functions providing Hastings polynomial solutions for a set of trigonometric functions. This package is designed to handle units of radians.

3.6.8.8.9.7.8.1.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Hastings_Radian_Operations | This package partially fulfills R217 |

3.6.8.8.9.7.8.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Radians | Floating point | Allows floating point representation of radian measurements. |
| Real | Floating point | General floating point representation. |
| Sin_Cos_Ratio | Floating point | Represents sines and cosines. |
| Tan_Ratio | Floating point | Represents tangent values. |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Pi_Over_2 | Radians | constant | constant value of Pi divided by 2 |
| Pi_Over_4 | Radians | constant | constant value of Pi divided by 4 |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply radians * radians yielding a real result. |

3.6.8.8.9.7.8.1.3 LOCAL ENTITIES

None.

3.6.8.8.9.7.8.1.4 INTERRUPTS

None.

3.6.8.8.9.7.8.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;
```

```
procedure Sample is
   type Angles   is digits 6;
   type FPs      is digits 6;
   type Sines    is digits 6;
   type Tangents is digits 6;

   This example shows these constants defined by the user.
   They are also available through the package Universal_Constants
   and may be used by with'ing that package.

   Pi          : constant := 3.14159;
   Pi_Over_2   : constant := Pi/2.0;
   Pi_Over_4   : constant := Pi/4.0;

   Right_Angle : Angle;
   Sine_Result : Sines;

   function "*" ( Left_Side  : Angle;
                  Right_Side : Angle) return FPs;

   package Hastings_Rad is new Polynomials.Hastings_Radian_Operations
                                   ( Radians       => Angles,
                                     Real          => FPs,
                                     Sin_Cos_Ratio => Sines,
                                     Tan_Ratio     => Tangents,
                                     Pi_Over_2     => Pi_Over_2,
                                     Pi_Over_4     => Pi_Over_4,
                                     Pi            => Pi,
                                     *             => * );
   begin
      Right_Angle := Pi_Over_2;
      Sine_Result := Hastings_Rad.Sin_R_5term( Right_Angle );
   end Sample;
```

3.6.8.8.9.7.8.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.7.8.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Sin_R_5term | function | Returns the sine of an angle computed with 5 terms, single or extended precision. |
| Sin_R_4term | function | Returns the sine of an angle computed with 4 terms, single or extended precision. |
| Cos_R_5term | function | Returns the cosine of an angle computed with 5 terms, single or extended precision. |
| Cos_R_4term | function | Returns the cosine of an angle computed with 4 terms, single or extended precision. |
| Tan_R_5term | function | Returns the tangent of an angle computed with 5 terms, single or extended precision. |
| Tan_R_4term | function | Returns the tangent of an angle computed with 4 terms, single or extended precision. |
| Arctan_R_8term | function | Returns an angle from the tangent computed with 8 terms, single or extended precision. |
| Arctan_R_7term | function | Returns an angle from the tangent computed with 7 terms, single or extended precision. |
| Arctan_R_6term | function | Returns an angle from the tangent computed with 6 terms, single or extended precision. |
| Mod_Arctan_R_8term | function | Returns an angle from the tangent computed with 8 terms, single or extended precision. |
| Mod_Arctan_R_7term | function | Returns an angle from the tangent computed with 7 terms, single or extended precision. |
| Mod_Arctan_R_6term | function | Returns an angle from the tangent computed with 6 terms, single or extended precision. |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Type | Catalog # |
|---|---|---|
| Sin_R_5term | function | P915-0 |
| Sin_R_4term | function | P916-0 |
| Cos_R_5term | function | P917-0 |
| Cos_R_4term | function | P918-0 |
| Tan_R_5term | function | P919-0 |
| Tan_R_4term | function | P920-0 |
| Arctan_R_8term | function | P921-0 |
| Arctan_R_7term | function | P922-0 |
| Arctan_R_6term | function | P923-0 |
| Mod_Arctan_R_8term | function | P924-0 |
| Mod_Arctan_R_7term | function | P925-0 |
| Mod_Arctan_R_6term | function | P926-0 |

3.6.8.8.9.7.8.1.8 PART DESIGN

None.

3.6.8.8.9.7.8.2 HASTINGS_DEGREE_OPERATIONS (CATALOG #P927-0)

This generic package contains generic functions providing Hastings polynomial solutions for a set of trigonometric functions. This package is designed to handle units of degrees.

3.6.8.8.9.7.8.2.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Hastings_Degree_Operations | This package partially fulfills R217 |

3.6.8.8.9.7.8.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Degrees | Floating point | Allows floating point representation of degree measurements. |
| Real | Floating point | General floating point representation. |
| Sin_Cos_Ratio | Floating point | Represents sines and cosines. |
| Tan_Ratio | Floating point | Represents tangent values. |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Pi | Degrees | constant | constant value of Pi |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply degrees * degrees yielding a real result. |

3.6.8.8.9.7.8.2.3 LOCAL ENTITIES

None.

3.6.8.8.9.7.8.2.4 INTERRUPTS

None.

3.6.8.8.9.7.8.2.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;
```

```
procedure Sample is
   type Angles   is digits 6;
   type FPs      is digits 6;
   type Sines    is digits 6;
   type Tangents is digits 6;

   Right_Angle : Angle;
   Sine_Result : Sines;

   function "*" ( Left_Side  : Angle;
                  Right_Side : Angle) return FPs;

   package Hast_Deg is new Polynomials.Hastings_Degree_Operations
                                   ( Degrees       => Angles,
                                     Real          => FPs,
                                     Sin_Cos_Ratio => Sines,
                                     Tan_Ratio     => Tangents,
                                     *             => * );
   begin
      Right_Angle := 90.0;
      Sine_Result := Hast_Deg.Sin_R_5term( Right_Angle );
   end Sample;
```

### 3.6.8.8.9.7.8.2.6 GLOBAL PROCESSING

**There is no global processing performed by this LLCSC.**

### 3.6.8.8.9.7.8.2.7 DECOMPOSITION

**The following table describes the decomposition of this part:**

| Name | Type | Description |
|---|---|---|
| Sin_D_5term | function | Returns the sine of an angle, computed with 5 terms, single or extended precision. |
| Sin_D_4term | function | Returns the sine of an angle computed with 4 terms, single or extended precision. |
| Cos_D_5term | function | Returns the cosine of an angle computed with 5 terms, single or extended precision. |
| Cos_D_4term | function | Returns the cosine of an angle computed with 4 terms, single or extended precision. |
| Tan_D_5term | function | Returns the tangent of an angle computed with 5 terms, single or extended precision. |
| Tan_D_4term | function | Returns the tangent of an angle computed with 4 terms, single or extended precision. |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Type | Catalog # |
|---|---|---|
| Sin_D_5term | function | P928-0 |
| Sin_D_4term | function | P929-0 |
| Cos_D_5term | function | P930-0 |
| Cos_D_4term | function | P931-0 |
| Tan_D_5term | function | P932-0 |
| Tan_D_4term | function | P933-0 |

3.6.8.8.9.7.8.2.8 PART DESIGN

None.

3.6.8.8.9.8 MODIFIED_NEWTON_RAPHSON (CATALOG #P934-0)

This package contains generic functions providing Modified Newton Raphson polynomial solutions for the square root function.

3.6.8.8.9.8.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Modified_Newton_Raphson | R220 |

3.6.8.8.9.8.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by the functions (Sqrt) in this part:

| Name | Type | Description |
|---|---|---|
| Inputs | Floating point | Floating point Input to square root function. |
| Outputs | Floating point | Floating point Output of square root function. |

3.6.8.8.9.8.3 LOCAL ENTITIES

None.

3.6.8.8.9.8.4 INTERRUPTS

None.

3.6.8.8.9.8.5 TIMING AND SEQUENCING

The following shows a sample usage of the only function in this part:

```
with Polynomials;

procedure Sample is
   type FPs      is digits 6;

   Sample_Num : FPs;
   Result     : FPs;

   function Mod_Sqrt is new Polynomials.Modified_Newton_Raphson.Sqrt
                                    ( Inputs => FPs,
                                      Outputs => FPs);
   begin
      Sample_Num := 642.33;
      Result := Mod_Sqrt( Input => Sample_Num );
   end Sample;
```

3.6.8.8.9.8.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.8.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Sqrt | generic function | Function returning the square root of a real. Calculation cycle performed 3 times. |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Catalog # |
|---|---|
| Sqrt | P935-0 |

3.6.8.8.9.8.8 PART DESIGN

None.

3.6.8.8.9.9 NEWTON_RAPHSON (CATALOG #P936-0)

This packages contains generic functions providing Newton Raphson polynomial solutions for the square root function.

3.6.8.8.9.9.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Newton_Raphson | R221 |

3.6.8.8.9.9.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by the only function (Sqrt) in this part:

| Name | Type | Description |
|---|---|---|
| Inputs | Floating point | Floating point Input to square root function. |
| Outputs | Floating point | Floating point Output of square root function. |

3.6.8.8.9.9.3 LOCAL ENTITIES

None.

3.6.8.8.9.9.4 INTERRUPTS

None.

3.6.8.8.9.9.5 TIMING AND SEQUENCING

The following shows a sample usage of the only function in this part:

```
with Polynomials;

procedure Sample is
   type FPs      is digits 9;

   Sample_Num : FPs;
   Result     : FPs;

   function Sqrt is new Polynomials.Newton_Raphson.Sqrt
                                   ( Inputs => FPs,
                                     Outputs => FPs);
   begin
      Sample_Num := 642.33;
      Result := Sqrt( Input => Sample_Num );
   end Sample;
```

3.6.8.8.9.9.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.9.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Sqrt | generic function | Function returning the square root of a real. Calculation cycle performed 3 times. |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Catalog # |
|---|---|
| Sqrt | P937-0 |

3.6.8.8.9.9.8 PART DESIGN

None.

### 3.6.8.8.9.10 SYSTEM_FUNCTIONS (CATALOG #P938-0)

This package provides access to the Ada system library for standard mathematical functions. The trigonometric functions allow for inputs with units of radians, semicircles, and degrees.

#### 3.6.8.8.9.10.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| System_Functions | R223 |

#### 3.6.8.8.9.10.2 INPUT/OUTPUT

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following chart describes the exceptions exported by this part:

| Name | Description |
|---|---|
| Invalid_Operand | The input value is in an improper format not accepted by the operating system |
| Invalid_Argument | The input value is an a range unacceptable to the function being called |
| Overflow | A floating point overflow was encountered during the calculations |
| Underflow | A floating point underflow was encountered during the calculations |
| Log_Zero_Negative | An attempt was made to take a log of a zero or negative value value |
| Square_Root_Negative | An attempt was made to take the square root of a negative number |

#### 3.6.8.8.9.10.3 LOCAL ENTITIES

None.

#### 3.6.8.8.9.10.4 INTERRUPTS

None.

3.6.8.8.9.10.5 TIMING AND SEQUENCING

None.

3.6.8.8.9.10.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.10.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Radian_Operations | generic package | Contains trigonometric functions dealing with units of radians |
| Semicircle_Operations | generic package | Contains trigonometric functions dealing with units of semicircles |
| Degree_Operations | generic package | Contains trigonometric functions dealing with units of degrees |
| Square_Root | generic | Contains a square root function |
| Base_10_Logarithm | generic | Contains a base 10 logarithm function |
| Base_N_Logarithm | generic | Contains a base n logarithm function |

3.6.8.8.9.10.8 PART DESIGN

3.6.8.8.9.10.8.1 RADIAN_OPERATIONS (CATALOG #P939-0)

Provides a set of trigonometric functions handling angles in units of radians.

No exceptions are raised by this part. The following exceptions are raised by units in this part:

| Name | Invalid Operand | Invalid Argument | Over-Flow |
|---|---|---|---|
| Sin | * | | |
| Cos | * | | |
| Tan | * | | * |
| Arcsin | * | * | |
| Arccos | * | * | |
| Arctan | * | | |

3.6.8.8.9.10.8.1.1 REQUIREMENTS ALLOCATION

None.

3.6.8.8.9.10.8.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Radians | floating point type | Data type describing units of angles |
| Sin_Cos_Ratio | floating point type | Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions |
| Tan_Ratio | floating point type | Data type describing output values from tangent function and input values to arctangent function |

3.6.8.8.9.10.8.1.3 LOCAL ENTITIES

None.

3.6.8.8.9.10.8.1.4 INTERRUPTS

None.

3.6.8.8.9.10.8.1.5 TIMING AND SEQUENCING

The following illustrates a sample usage of this part:

```
with Polynomials;
...
   type My_Radians is new FLOAT;
   type My_Sin_Cos_Ratio is new FLOAT;
   type My_Tan_Ratio     is new FLOAT;
   ...
   package ROpns is new
           Polynomials.System_Functions.Radian_Operations
              (Radians       => My_Radians,
               Sin_Cos_Ratio => My_Sin_Cos_Ratio,
               Tan_Ratio     => My_Tan_Ratio);
   ...
   Angle  : My_Radians;
   Result : My_Sin_Cos_Ratio;
   ...
   begin
      ...
      Result := ROpns.Sin(Angle);
```

3.6.8.8.9.10.8.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.10.8.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Sin | function | Sine function |
| Cos | function | Cosine function |
| Tan | function | Tangent function |
| Arcsin | function | Arcsine function |
| Arccos | function | Arccosine function |
| Arctan | function | Arctangent function |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Catalog # |
|---|---|
| Sin | P940-0 |
| Cos | P941-0 |
| Tan | P942-0 |
| Arcsin | P943-0 |
| Arccos | P944-0 |
| Arctan | P945-0 |

3.6.8.8.9.10.8.1.8 PART DESIGN

None.

3.6.8.8.9.10.8.2 SEMICIRCLE_OPERATIONS (CATALOG #P946-0)

Provides a set of trigonometric functions handling angles in units of semicircles.

No exceptions are raised by this part. The following exceptions are raised by units in this part:

| Name | Invalid Operand | Invalid Argument | Over-Flow |
|---|---|---|---|
| Sin | * | | |
| Cos | * | | |
| Tan | * | | * |
| Arcsin | * | * | |
| Arccos | * | * | |
| Arctan | * | | |

3.6.8.8.9.10.8.2.1 REQUIREMENTS ALLOCATION

See top header.

3.6.8.8.9.10.8.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Scalars | floating point type | Describes data type of input object pi |
| Semicircles | floating point type | Data type describing units of angles |
| Sin_Cos_Ratio | floating point type | Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions |
| Tan_Ratio | floating point type | Data type describing output values from tangent function and input values to arctangent function |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Pi | Scalars | N/A | Number of radians in a semicircle |

Subprograms:

The following table describes the generic formal subroutines (operators) required by this part:

| Name | Left Input Type | Right Input Type | Result Type |
|---|---|---|---|
| "*" | Semicircles | Scalars | Scalars |
| "*" | Scalars | Scalars | Semicircles |

3.6.8.8.9.10.8.2.3 LOCAL ENTITIES

None.

3.6.8.8.9.10.8.2.4 INTERRUPTS

None.

3.6.8.8.9.10.8.2.5 TIMING AND SEQUENCING

The following illustrates a sample usage of this part:

```
with Polynomials;
...
   type My_Semicircles   is new FLOAT;
   type My_Sin_Cos_Ratio is new FLOAT;
   type My_Tan_Ratio     is new FLOAT;
   ...
   function "*" (Left  : My_Semicircles;
                 Right : FLOAT) return FLOAT;
   function "*" (Left  : FLOAT;
                 Right : FLOAT) return My_Semicircles;
   ...
   package SOpns is new
           Polynomials.System_Functions.Semicircle_Operations
              (Scalars       => FLOAT,
               Semicircles   => My_Semicircles,
               Sin_Cos_Ratio => My_Sin_Cos_Ratio,
               Tan_Ratio     => My_Tan_Ratio);
   ...
   Angle  : My_Semicircles;
   Result : My_Sin_Cos_Ratio;
   ...
   begin
      ...
      Result := SOpns.Sin(Angle);
```

3.6.8.8.9.10.8.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.10.8.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Sin | function | Sine function |
| Cos | function | Cosine function |
| Tan | function | Tangent function |
| Arcsin | function | Arcsine function |
| Arccos | function | Arccosine function |
| Arctan | function | Arctangent function |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Catalog # |
|---|---|
| Sin | P947-0 |
| Cos | P948-0 |
| Tan | P949-0 |
| Arcsin | P950-0 |
| Arccos | P951-0 |
| Arctan | P952-0 |

3.6.8.8.9.10.8.2.8 PART DESIGN

None.

3.6.8.8.9.10.8.3 DEGREE_OPERATIONS (CATALOG #P953-0)

Provides a set of trigonometric functions handling angles in units of degrees.

No exceptions are raised by this part. The following exceptions are raised by units in this part:

| Name | Invalid Operand | Invalid Argument | Over-Flow | Under-Flow |
|---|---|---|---|---|
| Sin | * | | | * |
| Cos | * | | | |
| Tan | * | | * | |
| Arcsin | * | * | | |
| Arccos | * | * | | |
| Arctan | * | | | |

3.6.8.8.9.10.8.3.1 REQUIREMENTS ALLOCATION

See top header.

3.6.8.8.9.10.8.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Degrees | floating point type | Data type describing units of angles |
| Sin_Cos_Ratio | floating point type | Data type describing output values from sine and cosine functions and input values to arcsine and arccosine functions |
| Tan_Ratio | floating point type | Data type describing output values from tangent function and input values to arctangent function |

3.6.8.8.9.10.8.3.3 LOCAL ENTITIES

None.

3.6.8.8.9.10.8.3.4 INTERRUPTS

None.

3.6.8.8.9.10.8.3.5 TIMING AND SEQUENCING

The following illustrates a sample usage of this part:

```
with Polynomials;
...
   type My_Degrees   is new FLOAT;
   type My_Sin_Cos_Ratio is new FLOAT;
   type My_Tan_Ratio     is new FLOAT;
   ...
   package DOpns is new
           Polynomials.System_Functions.Degree_Operations
              (Degrees       => My_Degrees,
               Sin_Cos_Ratio => My_Sin_Cos_Ratio,
               Tan_Ratio     => My_Tan_Ratio);
   ...
   Angle  : My_Degrees;
   Result : My_Sin_Cos_Ratio;
   ...
   begin
```

```
...
Result := DOpns.Sin(Angle);
```

3.6.8.8.9.10.8.3.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.10.8.3.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Sin | function | Sine function |
| Cos | function | Cosine function |
| Tan | function | Tangent function |
| Arcsin | function | Arcsine function |
| Arccos | function | Arccosine function |
| Arctan | function | Arctangent function |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Catalog # |
|---|---|
| Sin | P954-0 |
| Cos | P955-0 |
| Tan | P956-0 |
| Arcsin | P957-0 |
| Arccos | P958-0 |
| Arctan | P959-0 |

3.6.8.8.9.10.8.3.8 PART DESIGN

None.

3.6.8.8.9.10.8.4 SQUARE_ROOT (CATALOG #P960-0)

This package contains the function required to calculated the square root of an input value.

The following exceptions are raised by units in this part:

| Name | Invalid Operand | Square_ Root_Negative |
|---|---|---|
| Sqrt | * | * |

3.6.8.8.9.10.8.4.1 REQUIREMENTS ALLOCATION

See top header.

3.6.8.8.9.10.8.4.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Inputs | floating point type | Data type of input values |
| Outputs | floating | Data type of output values |

3.6.8.8.9.10.8.4.3 LOCAL ENTITIES

None.

3.6.8.8.9.10.8.4.4 INTERRUPTS

None.

3.6.8.8.9.10.8.4.5 TIMING AND SEQUENCING

The following illustrates how this part would be used:

```
with Polynomials;
...
   type My_Type         is new FLOAT;
   type My_Type_Squared is new My_Type;
   ...
   package Square_Root is new
           Polynomials.System_Functions.Square_Root
              (Inputs  => My_Type_Squared,
               Outputs => My_Type);
   use Square_Root;
   ...
   a : My_Type;
   b : My_Type_Squared;
   ...
   begin
      ...
      A := Sqrt(B);
```

3.6.8.8.9.10.8.4.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.10.8.4.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Sqrt | function | Calculates the square root of an input value |

3.6.8.8.9.10.8.4.8 PART DESIGN

None.

3.6.8.8.9.10.8.5 BASE_10_LOGARITHM (CATALOG #P961-0)

This package contains the function which calculates the base-10 log of an input value.

The following exceptions are raised by units in this part:

| Name | Invalid Operand | Log_Zero_ Negative |
|---|---|---|
| Log_10 | * | * |

3.6.8.8.9.10.8.5.1 REQUIREMENTS ALLOCATION

See top header.

3.6.8.8.9.10.8.5.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Inputs | floating point type | Data type of input values |
| Outputs | floating | Data type of output values |

3.6.8.8.9.10.8.5.3 LOCAL ENTITIES

None.

3.6.8.8.9.10.8.5.4 INTERRUPTS

None.

3.6.8.8.9.10.8.5.5 TIMING AND SEQUENCING

The following shows a sample usage of this chart:

```
with Polynomials;
...
   type My_Type        is new FLOAT;
   type Log_10_Results is new FLOAT;
   ...
   package Base_10_Log is new
           Polynomials.System_Functions.Base_10_Logarithms
              (Inputs  => My_Type,
               Outputs => Log_10_Results);
   use Base_10_log;
   ...
   a : My_Type;
   b : Log_10_Results;
   ...
   begin
      B := Log_10(A);
```

3.6.8.8.9.10.8.5.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.10.8.5.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Log_10 | function | Returns the base 10 log of an input value |

3.6.8.8.9.10.8.5.8 PART DESIGN

None.

### 3.6.8.8.9.10.8.6 BASE_N_LOGARITHM (CATALOG #P962-0)

This package contains the function required to calculated the base n logarithm of an input value.

The following exceptions are raised by this part:

| Name | When/Why Raised |
|---|---|
| Invalid_Operand | Raised if the format of the value of Base_N is invalid and not accepted by the operating system |
| Log_Zero_Negative | Raised if the value of Base_N is not greater than 0 |

#### 3.6.8.8.9.10.8.6.1 REQUIREMENTS ALLOCATION

This part partially meets CAMP requirement R223.

#### 3.6.8.8.9.10.8.6.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Inputs | floating point type | Data type of input values |
| Outputs | floating point type | Data type of output values |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Base_N | POSITIVE | N/A | Determines the root of the logarithm |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Left Input Type | Right Input Type | Result Type |
|---|---|---|---|
| "*" | Outputs | Outputs | Outputs |
| "/" | Inputs | Inputs | Outputs |

3.6.8.8.9.10.8.6.3 LOCAL ENTITIES

None.

3.6.8.8.9.10.8.6.4 INTERRUPTS

None.

3.6.8.8.9.10.8.6.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;
...
   type My_Type        is new FLOAT;
   type My_Log_Results is new FLOAT;
   ...
   My_Log_Base : constant POSITIVE := 3;
   ...
   package Base_3_Log is new
           Polynomials.System_Functions.Base_N_Logarithm
              (Inputs  => My_Type,
               Outputs => My_Log_Results,
               Base_N  => My_Log_Base);
   ...
   function Log_3 (Input : My_Type) return My_Log_Results
                                    renames Base_3_Log.Log_N;
   ...
   a : My_Type;
   b : My_Log_Results;
   ...
   begin
      ...
      B := Log_3(A);
```

3.6.8.8.9.10.8.6.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.10.8.6.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Log_N | function | Calculates the base n logarithm of an input value |

3.6.8.8.9.10.8.6.8 PART DESIGN

None.

3.6.8.8.9.11 TAYLOR (CATALOG #P963-0)

This package contains generic packages providing Taylor and Modified Taylor polynomial solutions for a set of trigonometric functions. Provisions are made for the trigonometric functions to handle units of radians or degrees.

3.6.8.8.9.11.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Taylor | R222 |

3.6.8.8.9.11.2 INPUT/OUTPUT

None.

3.6.8.8.9.11.3 LOCAL ENTITIES

None.

3.6.8.8.9.11.4 INTERRUPTS

None.

3.6.8.8.9.11.5 TIMING AND SEQUENCING

None.

3.6.8.8.9.11.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.11.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Taylor_ Radian_ Operations | generic package | Trigonometric functions dealing with input in units of radians |
| Taylor_ Degree_ Operations | generic package | Trigonometric functions dealing with input in units of degrees |
| Taylor_ Natural_ Log | generic package | Natural log functions with floating point input and output |
| Taylor_ Log_ Base_N | generic package | Log functions for different bases with floating point input and output |

3.6.8.8.9.11.8 PART DESIGN

3.6.8.8.9.11.8.1 TAYLOR_RADIAN_OPERATIONS (CATALOG #P964-0)

This generic package contains functions providing Taylor and Modified Taylor polynomial solutions for a set of trigonometric functions. This package is designed to handle units of radians.

3.6.8.8.9.11.8.1.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Taylor_Radian_Operations | This package partially fulfills R222 |

3.6.8.8.9.11.8.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Radians | Floating point | Allows floating point representation of radian measurements. |
| Real | Floating point | General floating point representation. |
| Sin_Cos_Ratio | Floating point | Represents sines and cosines. |
| Tan_Ratio | Floating point | Represents tangent values. |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Pi_Over_2 | Radians | constant | constant value Pi divided by 2 |
| Pi_Over_4 | Radians | constant | constant value Pi divided by 4 |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply radians * radians yielding a real result. |

3.6.8.8.9.11.8.1.3 LOCAL ENTITIES

None.

3.6.8.8.9.11.8.1.4 INTERRUPTS

None.

3.6.8.8.9.11.8.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;

procedure Sample is
   type Angles   is digits 6;
   type FPs      is digits 6;
   type Sines    is digits 6;
   type Tangents is digits 6;
```

This example shows these constants defined by the user.

They are also available through the package Universal_Constants and may be used by with'ing that package.

```
Pi          : constant := 3.14159;
Pi_Over_2   : constant := 1.57079;
Pi_Over_4   : constant := 0.785398;

Right_Angle : Angle;
Sine_Result : Sines;

function "*" ( Left_Side  : Angle;
               Right_Side : Angle) return FPs;

package Taylor_Rad is new Polynomials.Taylor_Radian_Operations
                                    ( Radians       => Angles,
                                      Real          => FPs,
                                      Sin_Cos_Ratio => Sines,
                                      Tan_Ratio     => Tangents,
                                      Pi            => Pi,
                                      Pi_Over_2     => Pi_Over_2,
                                      Pi_Over_4     => Pi_Over_4,
                                      *             => * );
begin
   Right_Angle := Pi_Over_2;
   Sine_Result := Taylor_Rad.Sin_R_5term( Right_Angle );
end Sample;
```

### 3.6.8.8.9.11.8.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

### 3.6.8.8.9.11.8.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Sin_R_8term | function | Returns the sine of an angle computed with 8 terms, extended precision. |
| Sin_R_7term | function | Returns the sine of an angle computed with 7 terms, extended precision. |
| Sin_R_6term | function | Returns the sine of an angle computed with 6 terms, extended precision. |
| Sin_R_5term | function | Returns the sine of an angle computed with 5 terms, single or extended precision. |
| Sin_R_4term | function | Returns the sine of an angle computed with 4 terms, single precision. |
| Cos_R_8term | function | Returns the cosine of an angle computed with 8 terms, extended precision. |
| Cos_R_7term | function | Returns the cosine of an angle computed with 7 terms, extended precision. |
| Cos_R_6term | function | Returns the cosine of an angle computed with 6 terms, extended precision. |
| Cos_R_5term | function | Returns the cosine of an angle computed with 5 terms, single or extended precision. |
| Cos_R_4term | function | Returns the cosine of an angle computed with 4 terms, single precision. |
| Tan_R_8term | function | Returns the tangent of an angle computed with 8 terms, extended precision. |
| Arcsin_R_8term | function | Returns an angle from the sine computed with 8 terms, extended precision. |
| Arcsin_R_7term | function | Returns an angle from the sine computed with 7 terms, extended precision. |
| Arcsin_R_6term | function | Returns an angle from the sine computed with 6 terms, extended precision. |
| Arcsin_R_5term | function | Returns an angle from the sine computed with 5 terms, single or extended precision. |
| Arccos_R_8term | function | Returns an angle from the cosine computed with 8 terms, extended precision. |
| Arccos_R_7term | function | Returns an angle from the cosine computed with 7 terms, extended precision. |
| Arccos_R_6term | function | Returns an angle from the cosine computed with 6 terms, extended precision. |
| Arccos_R_5term | function | Returns an angle from the cosine computed with 5 terms, single or extended precision. |
| Arctan_R_8term | function | Returns an angle from the tangent computed with 8 terms, extended precision. |
| Arctan_R_7term | function | Returns an angle from the tangent computed with 7 terms, extended precision. |
| Arctan_R_6term | function | Returns an angle from the tangent computed with 6 terms, extended precision. |
| Arctan_R_5term | function | Returns an angle from the tangent computed with 5 terms, single or extended precision. |
| Arctan_R_4term | function | Returns an angle from the tangent computed with 4 terms, single precision. |
| Alt_Arctan_R_8term | function | Returns an angle from the tangent computed with 8 terms, extended precision. |
| Alt_Arctan_R_7term | function | Returns an angle from the tangent computed with 7 terms, extended precision. |
| Alt_Arctan_R_6term | function | Returns an angle from the tangent computed with 6 terms, extended precision. |

| | | |
|---|---|---|
| Alt_Arctan_R_5term | function | Returns an angle from the tangent computed with 5 terms, single or extended precision. |
| Alt_Arctan_R_4term | function | Returns an angle from the tangent computed with 4 terms, single precision. |
| Mod_Sin_R_8term | function | Returns the sine of an angle computed with 8 terms, extended precision. |
| Mod_Sin_R_7term | function | Returns the sine of an angle computed with 7 terms, extended precision. |
| Mod_Sin_R_6term | function | Returns the sine of an angle computed with 6 terms, extended precision. |
| Mod_Sin_R_5term | function | Returns the sine of an angle computed with 5 terms, single or extended precision. |
| Mod_Sin_R_4term | function | Returns the sine of an angle computed with 4 terms, single precision. |
| Mod_Cos_R_8term | function | Returns the cosine of an angle computed with 8 terms, extended precision. |
| Mod_Cos_R_7term | function | Returns the cosine of an angle computed with 7 terms, extended precision. |
| Mod_Cos_R_6term | function | Returns the cosine of an angle computed with 6 terms, extended precision. |
| Mod_Cos_R_5term | function | Returns the cosine of an angle computed with 5 terms, single or extended precision. |
| Mod_Cos_R_4term | function | Returns the cosine of an angle computed with 4 terms, single precision. |
| Mod_Tan_R_8term | function | Returns the tangent of an angle computed with 8 terms, extended precision. |
| Mod_Tan_R_7term | function | Returns the tangent of an angle computed with 7 terms, extended precision. |
| Mod_Tan_R_6term | function | Returns the tangent of an angle computed with 6 terms, extended precision. |
| Mod_Tan_R_5term | function | Returns the tangent of an angle computed with 5 terms, extended precision. |
| Mod_Tan_R_4term | function | Returns the tangent of an angle computed with 4 terms, extended precision. |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Catalog # |
|---|---|
| Sin_R_8term | P965-0 |
| Sin_R_7term | P966-0 |
| Sin_R_6term | P967-0 |
| Sin_R_5term | P968-0 |
| Sin_R_4term | P969-0 |
| Cos_R_8term | P970-0 |
| Cos_R_7term | P971-0 |
| Cos_R_6term | P972-0 |
| Cos_R_5term | P973-0 |
| Cos_R_4term | P974-0 |
| Tan_R_8term | P975-0 |
| Arcsin_R_8term | P976-0 |
| Arcsin_R_7term | P977-0 |
| Arcsin_R_6term | P978-0 |
| Arcsin_R_5term | P979-0 |
| Arccos_R_8term | P980-0 |
| Arccos_R_7term | P981-0 |
| Arccos_R_6term | P982-0 |
| Arccos_R_5term | P983-0 |
| Arctan_R_8term | P984-0 |
| Arctan_R_7term | P985-0 |
| Arctan_R_6term | P986-0 |
| Arctan_R_5term | P987-0 |
| Arctan_R_4term | P988-0 |
| Alt_Arctan_R_8term | P989-0 |
| Alt_Arctan_R_7term | P990-0 |
| Alt_Arctan_R_6term | P991-0 |

| | |
|---|---|
| Alt_Arctan_R_5term | P992-0 |
| Alt_Arctan_R_4term | P993-0 |
| Mod_Sin_R_8term | P994-0 |
| Mod_Sin_R_7term | P995-0 |
| Mod_Sin_R_6term | P996-0 |
| Mod_Sin_R_5term | P997-0 |
| Mod_Sin_R_4term | P998-0 |
| Mod_Cos_R_8term | P999-0 |
| Mod_Cos_R_7term | P1000-0 |
| Mod_Cos_R_6term | P1001-0 |
| Mod_Cos_R_5term | P1002-0 |
| Mod_Cos_R_4term | P1003-0 |
| Mod_Tan_R_8term | P1004-0 |
| Mod_Tan_R_7term | P1005-0 |
| Mod_Tan_R_6term | P1006-0 |
| Mod_Tan_R_5term | P1007-0 |
| Mod_Tan_R_4term | P1008-0 |

### 3.6.8.8.9.11.8.1.8 PART DESIGN

None.

### 3.6.8.8.9.11.8.2 TAYLOR_DEGREE_OPERATIONS (CATALOG #P1009-0)

This generic package contains functions providing Taylor and Modified Taylor polynomial solutions for a set of trigonometric functions. This package is designed to handle units of degrees.

### 3.6.8.8.9.11.8.2.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Taylor_Degree_Operations | This package partially fulfills R222 |

3.6.8.8.9.11.8.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Degrees | Floating point | Allows floating point representation of degree measurements. |
| Real | Floating point | General floating point representation. |
| Sin_Cos_Ratio | Floating point | Represents sines and cosines. |
| Tan_Ratio | Floating point | Represents tangent values. |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply degrees * degrees yielding a real result. |

3.6.8.8.9.11.8.2.3 LOCAL ENTITIES

None.

3.6.8.8.9.11.8.2.4 INTERRUPTS

None.

3.6.8.8.9.11.8.2.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;

procedure Sample is
   type Angles   is digits 6;
   type FPs      is digits 6;
```

```
type Sines    is digits 6;
type Tangents is digits 6;

Right_Angle : Angle;
Sine_Result : Sines;

function "*" ( Left_Side  : Angle;
               Right_Side : Angle) return FPs;

package Taylor_Deg is new Polynomials.Taylor_Degree_Operations
                                    ( Degrees       => Angles,
                                      Real          => FPs,
                                      Sin_Cos_Ratio => Sines,
                                      Tan_Ratio     => Tangents,
                                      *             => * );
begin
   Right_Angle := 90.0;
   Sine_Result := Taylor_Deg.Sin_D_5term( Right_Angle );
end Sample;
```

### 3.6.8.8.9.11.8.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

### 3.6.8.8.9.11.8.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Sin_D_8term | function | Returns the sine of an angle computed with 8 terms, extended precision. |
| Sin_D_7term | function | Returns the sine of an angle computed with 7 terms, extended precision. |
| Sin_D_6term | function | Returns the sine of an angle computed with 6 terms, extended precision. |
| Sin_D_5term | function | Returns the sine of an angle computed with 5 terms, single or extended precision. |
| Sin_D_4term | function | Returns the sine of an angle computed with 4 terms, single precision. |
| Cos_D_8term | function | Returns the cosine of an angle computed with 8 terms, extended precision. |
| Cos_D_7term | function | Returns the cosine of an angle computed with 7 terms, extended precision. |
| Cos_D_6term | function | Returns the cosine of an angle computed with 6 terms, extended precision. |
| Cos_D_5term | function | Returns the cosine of an angle computed with 5 terms, single or extended precision. |
| Cos_D_4term | function | Returns the cosine of an angle computed with 4 terms, single precision. |
| Tan_D_8term | function | Returns the tangent of an angle computed with 8 terms, extended precision. |
| Mod_Sin_D_8term | function | Returns the sine of an angle computed with 8 terms, extended precision. |
| Mod_Sin_D_7term | function | Returns the sine of an angle computed with 7 terms, extended precision. |
| Mod_Sin_D_6term | function | Returns the sine of an angle computed with 6 terms, extended precision. |
| Mod_Sin_D_5term | function | Returns the sine of an angle computed with 5 terms, single or extended precision. |
| Mod_Sin_D_4term | function | Returns the sine of an angle computed with 4 terms, single precision. |
| Mod_Cos_D_8term | function | Returns the cosine of an angle computed with 8 terms, extended precision. |
| Mod_Cos_D_7term | function | Returns the cosine of an angle computed with 7 terms, extended precision. |
| Mod_Cos_D_6term | function | Returns the cosine of an angle computed with 6 terms, extended precision. |
| Mod_Cos_D_5term | function | Returns the cosine of an angle computed with 5 terms, single or extended precision. |
| Mod_Cos_D_4term | function | Returns the cosine of an angle computed with 4 terms, single precision. |
| Mod_Tan_D_8term | function | Returns the tangent of an angle computed with 8 terms, extended precision. |
| Mod_Tan_D_7term | function | Returns the tangent of an angle computed with 7 terms, extended precision. |
| Mod_Tan_D_6term | function | Returns the tangent of an angle computed with 6 terms, extended precision. |
| Mod_Tan_D_5term | function | Returns the tangent of an angle computed with 5 terms, extended precision. |
| Mod_Tan_D_4term | function | Returns the tangent of an angle computed with 4 terms, extended precision. |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Catalog # |
|---|---|
| Sin_D_8term | P1010-0 |
| Sin_D_7term | P1011-0 |
| Sin_D_6term | P1012-0 |
| Sin_D_5term | P1013-0 |
| Sin_D_4term | P1035-0 |
| Cos_D_8term | P1014-0 |
| Cos_D_7term | P1015-0 |
| Cos_D_6term | P1016-0 |
| Cos_D_5term | P1017-0 |
| Cos_D_4term | P1018-0 |
| Tan_D_8term | P1019-0 |
| Mod_Sin_D_8term | P1020-0 |
| Mod_Sin_D_7term | P1021-0 |
| Mod_Sin_D_6term | P1022-0 |
| Mod_Sin_D_5term | P1023-0 |
| Mod_Sin_D_4term | P1024-0 |
| Mod_Cos_D_8term | P1025-0 |
| Mod_Cos_D_7term | P1026-0 |
| Mod_Cos_D_6term | P1027-0 |
| Mod_Cos_D_5term | P1028-0 |
| Mod_Cos_D_4term | P1029-0 |
| Mod_Tan_D_8term | P1030-0 |
| Mod_Tan_D_7term | P1031-0 |
| Mod_Tan_D_6term | P1032-0 |
| Mod_Tan_D_5term | P1033-0 |
| Mod_Tan_D_4term | P1034-0 |

### 3.6.8.8.9.11.8.2.8 PART DESIGN

None.

### 3.6.8.8.9.11.8.3 TAYLOR_NATURAL_LOG (CATALOG #P1036-0)

This generic package contains functions providing Taylor polynomial solutions for the natural log function.

#### 3.6.8.8.9.11.8.3.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Taylor_Natural_Log | partial fulfillment of R222 |

3.6.8.8.9.11.8.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Inputs | Floating point | Floating point input to the function |
| Outputs | Floating point | Floating point output to the function |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply Inputs * Inputs yielding a result of type Outputs. |

3.6.8.8.9.11.8.3.3 LOCAL ENTITIES

None.

3.6.8.8.9.11.8.3.4 INTERRUPTS

None.

3.6.8.8.9.11.8.3.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;

procedure Sample is
   type FPs      is digits 9;

   Sample_Num : FPs;
   Result     : FPs;

   package Nat_Log is new Polynomials.Taylor_Natural_Log
                                          ( Inputs  => FPs;
                                            Outputs => FPs);
   begin
      Sample_Num := 33.0
      Result := Nat_Log.Nat_Log_8term( Sample_Num );
   end Sample;
```

3.6.8.8.9.11.8.3.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.11.8.3.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Nat_Log_8term | function | Returns the natural logarithm of a number computed with 8 terms, either precision |
| Nat_Log_7term | function | Returns the natural logarithm of a number computed with 7 terms, either precision |
| Nat_Log_6term | function | Returns the natural logarithm of a number computed with 6 terms, either precision |
| Nat_Log_5term | function | Returns the natural logarithm of a number computed with 5 terms, either precision |
| Nat_Log_4term | function | Returns the natural logarithm of a number computed with 4 terms, either precision |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Catalog # |
|---|---|
| Nat_Log_8term | P1037-0 |
| Nat_Log_7term | P1038-0 |
| Nat_Log_6term | P1039-0 |
| Nat_Log_5term | P1040-0 |
| Nat_Log_4term | P1041-0 |

3.6.8.8.9.11.8.3.8 PART DESIGN

None.

3.6.8.8.9.11.8.4 TAYLOR_LOG_BASE_N (CATALOG #P1042-0)

This generic package contains functions providing Taylor polynomial solutions for the log function for base N.

3.6.8.8.9.11.8.4.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Taylor_Log_Base_N | partial fulfillment of R222 |

3.6.8.8.9.11.8.4.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Inputs | Floating point | Floating point input to the function |
| Outputs | Floating point | Floating point output to the function |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Base_N | Positive | defualt = 10 | Base to operate in |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| "*" | function | Overloaded operator to multiply Inputs * Inputs yielding a result of type Outputs. |

3.6.8.8.9.11.8.4.3 LOCAL ENTITIES

None.

3.6.8.8.9.11.8.4.4 INTERRUPTS

None.

3.6.8.8.9.11.8.4.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Polynomials;

procedure Sample is
   type FPs      is digits 9;

   Base       : Positive;
   Sample_Num : FPs;
   Result     : FPs;

   package Log_Base_5 is new Polynomials.Taylor_Log_Base_N
                                          ( Inputs  => FPs;
                                            Outputs => FPs,
                                            Base_N  => Base );
   begin
      Base := 5;
      Sample_Num := 33.0
      Result := Nat_Log.Nat_Log_8term( Sample_Num );
   end Sample;
```

3.6.8.8.9.11.8.4.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.8.8.9.11.8.4.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Log_Base_N_8term | function | Returns the logarithm to Base of a number computed with 8 terms, either precision |
| Log_Base_N_7term | function | Returns the logarithm to Base of a number computed with 7 terms, either precision |
| Log_Base_N_6term | function | Returns the logarithm to Base of a number computed with 6 terms, either precision |
| Log_Base_N_5term | function | Returns the logarithm to Base of a number computed with 5 terms, either precision |
| Log_Base_N_4term | function | Returns the logarithm to Base of a number computed with 4 terms, either precision |

The following table lists the catalog numbers for the decomposition of this part:

| Name | Catalog # |
|---|---|
| Log_Base_N_8term | P1043-0 |
| Log_Base_N_7term | P1044-0 |
| Log_Base_N_6term | P1045-0 |
| Log_Base_N_5term | P1046-0 |
| Log_Base_N_4term | P1047-0 |

3.6.8.8.9.11.8.4.8 PART DESIGN

None.

(This page left intentionally blank.)

```
package Polynomials is
pragma PAGE;
   package Chebyshev is
   pragma PAGE;

      generic
         type Radians       is digits <>;
         type Real          is digits <>;
         type Sin_Cos_Ratio is digits <>;
         One_Over_Pi        : in Radians;
         with function "*" (Left  : Radians;
                            Right : Radians) return Real is <>;
      package Chebyshev_Radian_Operations is

--       -- Chebyshev sine radian functions

         function Sin_R_5term(Input       : Radians) return Sin_Cos_Ratio;

      end Chebyshev_Radian_Operations;

      pragma PAGE;
      generic
         type Degrees       is digits <>;
         type Real          is digits <>;
         type Sin_Cos_Ratio is digits <>;
         with function "*" (Left  : Degrees;
                            Right : Degrees)          return Real is <>;
      package Chebyshev_Degree_Operations is

--       -- Chebyshev sine degree functions

         function Sin_D_5term (Input : Degrees) return Sin_Cos_Ratio;

      end Chebyshev_Degree_Operations;


      pragma PAGE;
      generic
         type Real          is digits <>;
         type Semicircles   is digits <>;
         type Sin_Cos_Ratio is digits <>;
         with function "*" (Left  : Semicircles;
                            Right : Semicircles) return Real is <>;
      package Chebyshev_Semicircle_Operations is

--       -- Chebyshev sine semicircle functions

         function Sin_S_5term (Input : Semicircles) return Sin_Cos_Ratio;

      end Chebyshev_Semicircle_Operations;

   end Chebyshev;

pragma PAGE;
   package Cody_Waite is
   pragma PAGE;
      generic
```

```
         type Inputs        is digits <>;
         type Outputs       is digits <>;
         with function "*"  (Left  : Inputs;
                             Right : Inputs)  return Outputs is <>;
      package Cody_Natural_Log is

         function Nat_Log ( Input : Inputs ) return Outputs;

      end Cody_Natural_Log;

pragma PAGE;
      generic
         type Inputs        is digits <>;
         type Outputs       is digits <>;
         Base_N             : in POSITIVE := 10;
         with function "*"  (Left  : Inputs;
                             Right : Inputs)  return Outputs is <>;
      package Cody_Log_Base_N is

         package Log_Base_N is
            function Log_N ( Input : Inputs ) return Outputs;
         end Log_Base_N;
      end Cody_Log_Base_N;

   end Cody_Waite;

pragma PAGE;
   package Continued_Fractions is

      pragma PAGE;
      generic
         type Radians       is digits <>;
         type Tan_Ratio     is digits <>;
         Default_Term_Count : in POSITIVE;
         with function "*" (Left  : Radians;
                            Right : Radians) return Tan_Ratio is <>;
       package Continued_Radian_Operations is

--       -- Continued Fractions tangent radian functions

         function Tan_R (Input      : Radians;
                         Term_Count : POSITIVE := Default_Term_Count )
                                                  return Tan_Ratio;

--       -- Continued Fractions arctangent radian functions

         function Arctan_R (Input      : Tan_Ratio;
                            Term_Count : POSITIVE := Default_Term_Count )
                                                  return Radians;

       end Continued_Radian_Operations;

end Continued_Fractions;

pragma PAGE;
   package Fike is
   pragma PAGE;
```

```
      generic
         type Semicircles   is digits <>;
         type Sin_Cos_Ratio is digits <>;
         type Real          is digits <>;
         with function Sqrt(Input : Real) return Real;
      package Fike_Semicircle_Operations is

--       --arcsine functions

         function Arcsin_S_6term (Input : Sin_Cos_Ratio) return Semicircles;

--       --arccosine functions

         function Arccos_S_6term (Input : Sin_Cos_Ratio) return Semicircles;

      end Fike_Semicircle_Operations;

   end Fike;

pragma PAGE;
   generic
      type Inputs         is digits <>;
      type Results        is digits <>;
      Coefficient_Count   : in POSITIVE;
      with function "**" (Left  : Inputs;
                          Right : POSITIVE)    return Results is <>;
   package General_Polynomial is

      type Coefficient_Records is
              record
                Coefficient : Results;
                Power_Of_X  : NATURAL;
              end record;

      subtype Table_Dimension is INTEGER range 1 .. Coefficient_Count;

      Polynomial_Definition : array (Table_Dimension) of Coefficient_Records;

      function Polynomial (Input : Inputs) return Results;

   end General_Polynomial;

pragma PAGE;
   package Hart is

      pragma PAGE;
      generic
         type Radians       is digits <>;
         type Real          is digits <>;
         type Sin_Cos_Ratio is digits <>;
         Pi                 : Radians;
         Pi_Over_2          : Radians;
         with function "*" (Left  : Radians;
                            Right : Radians)          return Real is <>;
      package Hart_Radian_Operations is
```

```
--          -- Hart radian cosine functions

            function Cos_R_5term (Input : Radians) return Sin_Cos_Ratio;

         end Hart_Radian_Operations;


         pragma PAGE;
         generic
            type Degrees       is digits <>;
            type Real          is digits <>;
            type Sin_Cos_Ratio is digits <>;
            with function "*" (Left  : Degrees;
                               Right : Degrees)          return Real is <>;
         package Hart_Degree_Operations is

--          -- Hart degree cosine functions

            function Cos_D_5term (Input : Degrees) return Sin_Cos_Ratio;

         end Hart_Degree_Operations;

      end Hart;

pragma PAGE;
      package Hastings is
      pragma PAGE;

         generic
            type Radians       is digits <>;
            type Real          is digits <>;
            type Sin_Cos_Ratio is digits <>;
            type Tan_Ratio     is digits <>;
            Pi_Over_2          : in Radians;
            Pi_Over_4          : in Radians;
            Pi                 : in Radians;
            with function "*" (Left  : Radians;
                               Right : Radians)          return Real is <>;
         package Hastings_Radian_Operations is

--          -- Hastings sine radian functions

            function Sin_R_5term (Input : Radians) return Sin_Cos_Ratio;

            function Sin_R_4term (Input : Radians) return Sin_Cos_Ratio;

--          -- Hastings cosine radian functions

            function Cos_R_5term (Input : Radians) return Sin_Cos_Ratio;

            function Cos_R_4term (Input : Radians) return Sin_Cos_Ratio;

--          -- Hastings tangent radian functions

            function Tan_R_5term (Input : Radians) return Tan_Ratio;

            function Tan_R_4term (Input : Radians) return Tan_Ratio;
```

```
--          -- Hastings arctangent radian functions

            function Arctan_R_8term (Input : Tan_Ratio) return Radians;

            function Arctan_R_7term (Input : Tan_Ratio) return Radians;

            function Arctan_R_6term (Input : Tan_Ratio) return Radians;

--          -- Modified Hastings arctangent radian functions

            function Mod_Arctan_R_8term (Input : Tan_Ratio) return Radians;

            function Mod_Arctan_R_7term (Input : Tan_Ratio) return Radians;

            function Mod_Arctan_R_6term (Input : Tan_Ratio) return Radians;

         end Hastings_Radian_Operations;

pragma PAGE;
         generic
            type Degrees       is digits <>;
            type Real          is digits <>;
            type Sin_Cos_Ratio is digits <>;
            type Tan_Ratio     is digits <>;
            with function "*" (Left  : Degrees;
                               Right : Degrees)          return Real is <>;
         package Hastings_Degree_Operations is

--          -- Hastings sine degree functions

            function Sin_D_5term (Input : Degrees) return Sin_Cos_Ratio;

            function Sin_D_4term (Input : Degrees) return Sin_Cos_Ratio;

--          -- Hastings cosine degree functions

            function Cos_D_5term (Input : Degrees) return Sin_Cos_Ratio;

            function Cos_D_4term (Input : Degrees) return Sin_Cos_Ratio;

--          -- tangent degree functions

            function Tan_D_5term (Input : Degrees) return Tan_Ratio;

            function Tan_D_4term (Input : Degrees) return Tan_Ratio;

         end Hastings_Degree_Operations;

      end Hastings;

pragma PAGE;
      package Modified_Newton_Raphson is

--          -- miscellaneous function

         generic
```

```
         type Inputs        is digits <>;
         type Outputs       is digits <>;
      function Sqrt (Input : Inputs) return Outputs;

   end Modified_Newton_Raphson;

pragma PAGE;
   package Newton_Raphson is

--       -- miscellaneous function

      generic
         type Inputs        is digits <>;
         type Outputs       is digits <>;
      function Sqrt (Input : Inputs) return Outputs;

   end Newton_Raphson;

pragma PAGE;
   package System_Functions is

      Invalid_Operand      : exception;
      Invalid_Argument     : exception;
      Overflow             : exception;
      Underflow            : exception;
      Log_Zero_Negative    : exception;
      Square_Root_Negative : exception;

pragma PAGE;
      generic
         type Radians       is digits <>;
         type Sin_Cos_Ratio is digits <>;
         type Tan_Ratio     is digits <>;
      package Radian_Operations is

         function Sin    (Input : Radians)       return Sin_Cos_Ratio;
         function Cos    (Input : Radians)       return Sin_Cos_Ratio;
         function Tan    (Input : Radians)       return Tan_Ratio;
         function Arcsin (Input : Sin_Cos_Ratio) return Radians;
         function Arccos (Input : Sin_Cos_Ratio) return Radians;
         function Arctan (Input : Tan_Ratio)     return Radians;

      end Radian_Operations;

pragma PAGE;
      generic
         type Scalars       is digits <>;
         type Semicircles   is digits <>;
         type Sin_Cos_Ratio is digits <>;
         type Tan_Ratio     is digits <>;
         Pi                 : in Scalars;
         with function "*" (Left  : Semicircles;
                            Right : Scalars)       return Scalars is <>;
         with function "*" (Left  : Scalars;
                            Right : Scalars) return Semicircles is <>;
      package Semicircle_Operations is
```

```
      function Sin    (Input : Semicircles)   return Sin_Cos_Ratio;
      function Cos    (Input : Semicircles)   return Sin_Cos_Ratio;
      function Tan    (Input : Semicircles)   return Tan_Ratio;
      function Arcsin (Input : Sin_Cos_Ratio) return Semicircles;
      function Arccos (Input : Sin_Cos_Ratio) return Semicircles;
      function Arctan (Input : Tan_Ratio)     return Semicircles;

   end Semicircle_Operations;

pragma PAGE;
   generic
      type Degrees       is digits <>;
      type Sin_Cos_Ratio is digits <>;
      type Tan_Ratio     is digits <>;
   package Degree_Operations is

      function Sin    (Input : Degrees)       return Sin_Cos_Ratio;
      function Cos    (Input : Degrees)       return Sin_Cos_Ratio;
      function Tan    (Input : Degrees)       return Tan_Ratio;
      function Arcsin (Input : Sin_Cos_Ratio) return Degrees;
      function Arccos (Input : Sin_Cos_Ratio) return Degrees;
      function Arctan (Input : Tan_Ratio)     return Degrees;

   end Degree_Operations;

pragma PAGE;
   generic
      type Inputs  is digits <>;
      type Outputs is digits <>;
   package Square_Root is

      function Sqrt (Input : Inputs) return Outputs;

   end Square_Root;

pragma PAGE;
   generic
      type Inputs  is digits <>;
      type Outputs is digits <>;
   package Base_10_Logarithm is

      function Log_10 (Input : Inputs) return Outputs;

   end Base_10_Logarithm;

pragma PAGE;
   generic
      type Inputs  is digits <>;
      type Outputs is digits <>;
      Base_N       : in POSITIVE;
      with function "*" (Left  : Inputs;
                         Right : Inputs) return Outputs is <>;
      with function "/" (Left  : Inputs;
                         Right : Inputs) return Inputs is <>;
   package Base_N_Logarithm is

      function Log_N (Input : Inputs) return Outputs;
```

```
      end Base_N_Logarithm;

   end System_Functions;

pragma PAGE;
   package Taylor_Series is
   pragma PAGE;
      generic
         type Radians       is digits <>;
         type Real          is digits <>;
         type Sin_Cos_Ratio is digits <>;
         type Tan_Ratio     is digits <>;
         Pi                 : Radians;
         Pi_Over_2          : Radians;
         Pi_Over_4          : Radians;
         with function "*" (Left  : Radians;
                            Right : Radians) return Real is <>;
      package Taylor_Radian_Operations is

--       -- Taylor sine radian functions

         function Sin_R_8term (Input : Radians) return Sin_Cos_Ratio;

         function Sin_R_7term (Input : Radians) return Sin_Cos_Ratio;

         function Sin_R_6term (Input : Radians) return Sin_Cos_Ratio;

         function Sin_R_5term (Input : Radians) return Sin_Cos_Ratio;

         function Sin_R_4term (Input : Radians) return Sin_Cos_Ratio;

--       -- Modified Taylor sine radian functions

         function Mod_Sin_R_8term (Input : Radians) return Sin_Cos_Ratio;

         function Mod_Sin_R_7term (Input : Radians) return Sin_Cos_Ratio;

         function Mod_Sin_R_6term (Input : Radians) return Sin_Cos_Ratio;

         function Mod_Sin_R_5term (Input : Radians) return Sin_Cos_Ratio;

         function Mod_Sin_R_4term (Input : Radians) return Sin_Cos_Ratio;

--       -- Taylor cosine radian functions

         function Cos_R_8term(Input : Radians) return Sin_Cos_Ratio;

         function Cos_R_7term(Input : Radians) return Sin_Cos_Ratio;

         function Cos_R_6term(Input : Radians) return Sin_Cos_Ratio;

         function Cos_R_5term(Input : Radians) return Sin_Cos_Ratio;

         function Cos_R_4term(Input : Radians) return Sin_Cos_Ratio;

--       -- Modified Taylor cosine radian functions
```

```
        function Mod_Cos_R_8term(Input : Radians) return Sin_Cos_Ratio;
        function Mod_Cos_R_7term(Input : Radians) return Sin_Cos_Rat
        function Mod_Cos_R_6term(Input : Radians) return Sin_Cos_Ratio;
        function Mod_Cos_R_5term(Input : Radians) return Sin_Cos_Ratio;
        function Mod_Cos_R_4term(Input : Radians) return Sin_Cos_Ratio;
--      -- Taylor tangent radian functions
        function Tan_R_8term (Input : Radians) return Tan_Ratio;
--      -- Modified Taylor tangent functions
        function Mod_Tan_R_8term (Input : Radians) return Tan_Ratio;
        function Mod_Tan_R_7term (Input : Radians) return Tan_Ratio;
        function Mod_Tan_R_6term (Input : Radians) return Tan_Ratio;
        function Mod_Tan_R_5term (Input : Radians) return Tan_Ratio;
        function Mod_Tan_R_4term (Input : Radians) return Tan_Ratio;
--      -- Taylor arcsine radian functions
        function Arcsin_R_8term (Input : Sin_Cos_Ratio) return Radians;
        function Arcsin_R_7term (Input : Sin_Cos_Ratio) return Radians;
        function Arcsin_R_6term (Input : Sin_Cos_Ratio) return Radians;
        function Arcsin_R_5term (Input : Sin_Cos_Ratio) return Radians;
--      -- Taylor arccosine radian functions
        function Arccos_R_8term (Input : Sin_Cos_Ratio) return Radians;
        function Arccos_R_7term (Input : Sin_Cos_Ratio) return Radians;
        function Arccos_R_6term (Input : Sin_Cos_Ratio) return Radians;
        function Arccos_R_5term (Input : Sin_Cos_Ratio) return Radians;
--      --" Taylor arctangent radian functions
        function Arctan_R_8term (Input : Tan_Ratio) return Radians;
        function Arctan_R_7term (Input : Tan_Ratio) return Radians;
        function Arctan_R_6term (Input : Tan_Ratio) return Radians;
        function Arctan_R_5term (Input : Tan_Ratio) return Radians;
```

```
         function Arctan_R_4term (Input : Tan_Ratio) return Radians;

--       --" Alternate Taylor arctangent radian functions

         function Alt_Arctan_R_8term (Input : Tan_Ratio) return Radians;

         function Alt_Arctan_R_7term (Input : Tan_Ratio) return Radians;

         function Alt_Arctan_R_6term (Input : Tan_Ratio) return Radians;

         function Alt_Arctan_R_5term (Input : Tan_Ratio) return Radians;

         function Alt_Arctan_R_4term (Input : Tan_Ratio) return Radians;

      end Taylor_Radian_Operations;


pragma PAGE;
      generic
         type Degrees       is digits <>;
         type Real          is digits <>;
         type Sin_Cos_Ratio is digits <>;
         type Tan_Ratio     is digits <>;
         with function "*" (Left  : Degrees;
                            Right : Degrees)          return Real is <>;
      package Taylor_Degree_Operations is

--       --" Taylor sine degree functions

         function Sin_D_8term (Input : Degrees) return Sin_Cos_Ratio;

         function Sin_D_7term (Input : Degrees) return Sin_Cos_Ratio;

         function Sin_D_6term (Input : Degrees) return Sin_Cos_Ratio;

         function Sin_D_5term (Input : Degrees) return Sin_Cos_Ratio;

--       --" Modified Taylor sine degree functions

         function Mod_Sin_D_8term (Input : Degrees) return Sin_Cos_Ratio;

         function Mod_Sin_D_7term (Input : Degrees) return Sin_Cos_Ratio;

         function Mod_Sin_D_6term (Input : Degrees) return Sin_Cos_Ratio;

         function Mod_Sin_D_5term (Input : Degrees) return Sin_Cos_Ratio;

         function Mod_Sin_D_4term (Input : Degrees) return Sin_Cos_Ratio;

--       --" Taylor cosine degree functions

         function Cos_D_8term (Input : Degrees) return Sin_Cos_Ratio;

         function Cos_D_7term (Input : Degrees) return Sin_Cos_Ratio;

         function Cos_D_6term (Input : Degrees) return Sin_Cos_Ratio;
```

```
         function Cos_D_5term (Input : Degrees) return Sin_Cos_Ratio;

--       --" Modified Taylor cosine degree functions

         function Mod_Cos_D_8term (Input : Degrees) return Sin_Cos_Ratio;

         function Mod_Cos_D_7term (Input : Degrees) return Sin_Cos_Ratio;

         function Mod_Cos_D_6term (Input : Degrees) return Sin_Cos_Ratio;

         function Mod_Cos_D_5term (Input : Degrees) return Sin_Cos_Ratio;

         function Mod_Cos_D_4term (Input : Degrees) return Sin_Cos_Ratio;

--       --" Taylor tangent degree functions

         function Tan_D_8term (Input : Degrees) return Tan_Ratio;

--       --" Modified Taylor tangent degree functions

         function Mod_Tan_D_8term (Input : Degrees) return Tan_Ratio;

         function Mod_Tan_D_7term (Input : Degrees) return Tan_Ratio;

         function Mod_Tan_D_6term (Input : Degrees) return Tan_Ratio;

         function Mod_Tan_D_5term (Input : Degrees) return Tan_Ratio;

         function Mod_Tan_D_4term (Input : Degrees) return Tan_Ratio;

      end Taylor_Degree_Operations;

pragma PAGE;
      generic
         type Inputs        is digits <>;
         type Outputs       is digits <>;
         with function "*"  (Left  : Inputs;
                             Right : Inputs)  return Outputs is <>;
      package Taylor_Natural_Log is

         function Nat_Log_8term ( Input : Inputs ) return Outputs;

         function Nat_Log_7term ( Input : Inputs ) return Outputs;

         function Nat_Log_6term ( Input : Inputs ) return Outputs;

         function Nat_Log_5term ( Input : Inputs ) return Outputs;

         function Nat_Log_4term ( Input : Inputs ) return Outputs;

      end Taylor_Natural_Log;

pragma PAGE;
      generic
         type Inputs        is digits <>;
         type Outputs       is digits <>;
         Base_N             : in POSITIVE := 10;
```

```
      with function "*"  (Left  : Inputs;
                          Right : Inputs)  return Outputs is <>;
   package Taylor_Log_Base_N is

      package Log_Base_N_8term is
         function Log_N_8term ( Input : Inputs ) return Outputs;
      end Log_Base_N_8term;

      package Log_Base_N_7term is
         function Log_N_7term ( Input : Inputs ) return Outputs;
      end Log_Base_N_7term;

      package Log_Base_N_6term is
         function Log_N_6term ( Input : Inputs ) return Outputs;
      end Log_Base_N_6term;

      package Log_Base_N_5term is
         function Log_N_5term ( Input : Inputs ) return Outputs;
      end Log_Base_N_5term;

      package Log_Base_N_4term is
         function Log_N_4term ( Input : Inputs ) return Outputs;
      end Log_Base_N_4term;

   end Taylor_Log_Base_N;

end Taylor_Series;

end Polynomials;
```

### 3.6.8.9 QUATERNION_OPERATIONS (PACKAGE SPECIFICATION) TLCSC (CATALOG #P123-0)

This part, which is designed as an Ada package, contains specifications for all CAMP parts which can be used on Quaternions. These parts apply to missile navigation.

#### 3.6.8.9.1 REQUIREMENTS ALLOCATION

None.

#### 3.6.8.9.2 INPUT/OUTPUT

None.

#### 3.6.8.9.3 UTILIZATION OF OTHER ELEMENTS

None.

#### 3.6.8.9.4 LOCAL ENTITIES

None.

#### 3.6.8.9.5 INTERRUPTS

None.

#### 3.6.8.9.6 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Quaternion_Operations;
with Basic_Data_Types;
...
   package BDT_PKG   renames Basic_Data_Types;
   ...
   type Quat_Indices       is (Q0,Q1,Q2,Q3);
   type Quaternion_Vectors is array (Quat_Indices)
                                 of BDT_PKG.Trig.Sin_Cos_Ratio;

   function "*" (Left  : BDT_PKG.Trig.Sin_Cos_Ratio;
                 Right : FLOAT) return  BDT_PKG.Trig.Sin_Cos_Ratio;

   package Quat_PKG is new
                   Quaternion_Operations
                      (Quaternion_Indices => Quat_Indices,
                       Sin_Cos_Ratio      => BDT_PKG.Trig.Sin_Cos_Ratio,
                       Quaternion_Vectors => Quaternion_Vectors;
                       Real               => FLOAT);

   ...
```

3.6.8.9.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.8.9.8 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Quaternion_Computed_From_Euler_Angles | generic function | Computes the unit Quaternion that represents the orientation of one frame to another. |
| Normalized_Quaternion | generic function | Normalizes a Quaternion when applied repeatedly. |
| "*" | generic function | Computes the product of two Quaternions. |

3.6.8.9.9 PART DESIGN

3.6.8.9.9.1 QUATERNION_COMPUTED_FROM_EULER_ANGLES (CATALOG #P124-0)

This part computes the unit Quaternion, Q, that represents the orientation of frame xyz with respect to XYZ (i.e. Q rotates XYZ into xyz) given the Euler angles relating xyz to XYZ.

3.6.8.9.9.1.1 REQUIREMENTS ALLOCATION

None.

3.6.8.9.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Euler_Angle_Indices | discrete type | Data type representing the index to the vector Euler_Angle_Vectors which has values such as Psi, Theta, and Phi. |
| Angles | floating point type | Data type for the elements of the Euler angle vector. |
| Euler_Angle_Vectors | array | Data type representing the Euler angles. |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Psi | Euler_Angle_Indices | 'FIRST | This object is the first Euler angle rotation that rotates XYZ into X'Y'Z' by rotating XYZ thru the angle Psi about the Z-axis. |
| Theta | Euler_Angle_Indices | 'SUCC (Psi) | This object is the second Euler angle rotation that rotates X'Y'Z' into X''Y''Z'' by rotating X'Y'Z' thru the angle Theta about the Y'-axis. |
| Phi | Euler_Angle_Indices | 'LAST | This object is the third Euler angle rotation that rotates X''Y''Z'' into X'''Y'''Z''' by rotating X''Y''Z'' thru the angle Phi about the X''-axis. |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| Sin_Cos | procedure | Procedure returning the sine and cosine of an euler angle (of type "Angles") |
| "*" | function | Function multiplying a type Sin_Cos_Ratio by a type Real returning type Sin_Cos_Ratio. |

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

| Name | Type | Mode | Description |
|---|---|---|---|
| Euler_Angles | Euler_Angle_Vectors | In | This value is a vector representing the euler angles. |

3.6.8.9.9.1.3 INTERRUPTS

None.

3.6.8.9.9.1.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Quaternion_Operations;
with Basic_Data_Types;
...
   package BDT_PKG   renames Basic_Data_Types;
  ...
   type Quat_Indices       is (Q0,Q1,Q2,Q3);
   type Quaternion_Vectors is array (Quat_Indices)
                                 of BDT_PKG.Trig.Sin_Cos_Ratio;

   type Euler_Indices    is (psi, Theta, Phi);
   type Euler_Angles     is new BDT_PKG.Trig.Radians;
   type Euler_Vectors    is array (Euler_Indices) of Euler_Angles;

   function "*" (Left  : BDT_PKG.Trig.Sin_Cos_Ratio;
                 Right : FLOAT) return  BDT_PKG.Trig.Sin_Cos_Ratio;

   package Quat_PKG is new
                   Quaternion_Operations
                    (Quaternion_Indices => Quat_Indices,
                       Sin_Cos_Ratio      => BDT_PKG.Trig.Sin_Cos_Ratio,
                       Quaternion_Vectors => Quaternion_Vectors;
                       Real               => FLOAT);

   procedure Sin_Cos
             (value                :in     Euler_Angles;
              Sin_of_Wander_Angle :   out BDT_PKG.Trig.Sin_Cos_Ratio;
              Cos_of_Wander_Angle :   out BDT_PKG.Trig.Sin_Cos_Ratio);

   function Compute_Q_From_Euler_Angles is new
            Quat_PKG.Quaternion_Computed_From_Euler_Angles
               (Euler_Angle_Indices  => Euler_Indices,
                Angles               => Euler_Angles,
                Euler_Angle_Vectors  => Euler_Vectors);
   ...
   Quaternion        : Quaternion_Vectors;
   Euler_Angles      : Euler_Vectors;
   ...
```

```
begin
   ...
   Quaternion := Compute_Q_From_Euler_Angles (Euler_Angles);
```

3.6.8.9.9.1.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.9.9.1.6 DECOMPOSITION

None.

3.6.8.9.9.2 NORMALIZED_QUATERNION (CATALOG #P125-0)

This function normalizes a Quaternion when applied repeatedly. One iteration will not (in most cases) normalize the Quaternion. The frequency of execution is dependent upon the desired accuracy, the length of the time interval between updates, and other application-dependent factors. This part is usually applied repeatedly over time.

3.6.8.9.9.2.1 REQUIREMENTS ALLOCATION

None.

3.6.8.9.9.2.2 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

| Name | Type | Mode | Description |
|---|---|---|---|
| Quaternion | Quaternion_Vectors | In | This value is a vector representing a Quaternion vector. |

3.6.8.9.9.2.3 INTERRUPTS

None.

3.6.8.9.9.2.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Quaternion_Operations;
with Basic_Data_Types;
...
  package BDT_PKG   renames Basic_Data_Types;
   ...
```

```
type Quat_Indices       is (Q0,Q1,Q2,Q3);
type Quaternion_Vectors is array (Quat_Indices)
                                 of BDT_PKG.Trig.Sin_Cos_Ratio;
function "*" (Left  : BDT_PKG.Trig.Sin_Cos_Ratio;
              Right : FLOAT) return  BDT_PKG.Trig.Sin_Cos_Ratio;

package Quat_PKG is new
                Quaternion_Operations
                 (Quaternion_Indices => Quat_Indices,
                    Sin_Cos_Ratio      => BDT_PKG.Trig.Sin_Cos_Ratio,
                    Quaternion_Vectors => Quaternion_Vectors;
                    Real               => FLOAT);
...
Quaternion        : Quaternion_Vectors;
...
begin
   ...
   Quaternion := Quat_PKG.Normalized_Quaternion (Quaternion);
```

3.6.8.9.9.2.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.9.9.2.6 DECOMPOSITION

None.

3.6.8.9.9.3 "*" (CATALOG #P128-0)

This generic function computes the product of two Quaternions.

3.6.8.9.9.3.1 REQUIREMENTS ALLOCATION

None.

3.6.8.9.9.3.2 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

| Name | Type | Mode | Description |
|---|---|---|---|
| Quaternion_A | Quaternion_Vectors | In | This value is a vector representing a Quaternion vector. |
| Quaternion_B | Quaternion_Vectors | In | This value is a vector representing a Quaternion vector. |

3.6.8.9.9.3.3 INTERRUPTS

None.

3.6.8.9.9.3.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Quaternion_Operations;
with Basic_Data_Types;
...
   package BDT_PKG   renames Basic_Data_Types;
   ...
   type Quat_Indices       is (Q0,Q1,Q2,Q3);
   type Quaternion_Vectors is array (Quat_Indices)
                                 of BDT_PKG.Trig.Sin_Cos_Ratio;
   function "*" (Left  : BDT_PKG.Trig.Sin_Cos_Ratio;
                 Right : FLOAT) return  BDT_PKG.Trig.Sin_Cos_Ratio;

   package Quat_PKG is new
                   Quaternion_Operations
                    (Quaternion_Indices => Quat_Indices,
                       Sin_Cos_Ratio      => BDT_PKG.Trig.Sin_Cos_Ratio,
                       Quaternion_Vectors => Quaternion_Vectors;
                       Real               => FLOAT);
   ...
   Quaternion_A      : Quaternion_Vectors;
   Quaternion_B      : Quaternion_Vectors;
   Quaternion_C      : Quaternion_Vectors;
   ...
   begin
      ...
      Quaternion_C := Quaternion_A * Quaternion_B;
```

3.6.8.9.9.3.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.8.9.9.3.6 DECOMPOSITION

None.

**(This page left intentionally blank.)**

```
generic

   type Quaternion_Indices is (<>);
   type Sin_Cos_Ratio      is digits <>;
   type Quaternion_Vectors is array (Quaternion_Indices)
                                    of Sin_Cos_Ratio;
   type Real               is digits <>;

   with function "*" (Left  : Sin_Cos_Ratio;
                      Right : Real)
                     return Sin_Cos_Ratio is <>;

   Q0    : in Quaternion_Indices := Quaternion_Indices'FIRST;
   Q1    : in Quaternion_Indices := Quaternion_Indices'SUCC(Q0);
   Q2    : in Quaternion_Indices :=
                    Quaternion_Indices'SUCC(Quaternion_Indices'SUCC(Q0));
   Q3    : in Quaternion_Indices := Quaternion_Indices'LAST;

package Quaternion_Operations is


pragma PAGE;
     generic

        type Euler_Angle_Indices is (<>);
        type Angles              is digits <>;
        type Euler_Angle_Vectors is array (Euler_Angle_Indices)
                                          of Angles;

        Psi   : in Euler_Angle_Indices := Euler_Angle_Indices'FIRST;
        Theta : in Euler_Angle_Indices := Euler_Angle_Indices'SUCC(Psi);
        Phi   : in Euler_Angle_Indices := Euler_Angle_Indices'LAST;

        with procedure Sin_Cos (Input     : in     Angles;
                                Sin_Value :    out Sin_Cos_Ratio;
                                Cos_Value :    out Sin_Cos_Ratio) is <>;



     function Quaternion_Computed_From_Euler_Angles
                           (Euler_Angles  : Euler_Angle_Vectors)
                           return Quaternion_Vectors;

pragma PAGE;
   function Normalized_Quaternion (Quaternion : Quaternion_Vectors)
                                   return Quaternion_Vectors;

pragma PAGE;
    function "*" (Quaternion_A : Quaternion_Vectors;
                  Quaternion_B : Quaternion_Vectors)
                 return Quaternion_Vectors;


end Quaternion_Operations;
```

**(This page left intentionally blank.)**

### 3.6.9 ABSTRACT MECHANISMS

(This page left intentionally blank.)

### 3.6.9.1 ABSTRACT_DATA_STRUCTURES (PACKAGE SPECIFICATION) TLCSC (CATALOG #P323-0)

This package contains the generic packages required to define and manipulate the following abstract data structures:

- o bounded FIFO buffer
- o unbounded FIFO buffer
- o nonblocking circular buffer
- o unbounded priority queue
- o bounded stack
- o unbounded stack

#### 3.6.9.1.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Bounded_FIFO_Buffer | R125 |
| Unbounded_FIFO_Buffer | R164 |
| Nonblocking_Circular_Buffer | R126 |
| Unbounded_Priority_Queue | R165 |
| Bounded_Stack | R166 |
| Unbounded_Stack | R167 |

#### 3.6.9.1.2 INPUT/OUTPUT

None.

#### 3.6.9.1.3 UTILIZATION OF OTHER ELEMENTS

None.

#### 3.6.9.1.4 LOCAL ENTITIES

None.

#### 3.6.9.1.5 INTERRUPTS

None.

#### 3.6.9.1.6 TIMING AND SEQUENCING

None.

3.6.9.1.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.9.1.8 DECOMPOSITION

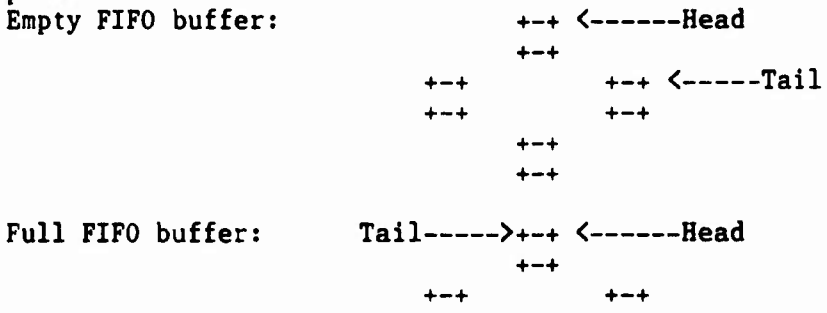
The following table describes the decomposition of this part:

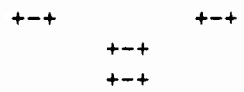
| Name | Type | Description |
|---|---|---|
| Bounded_FIFO_Buffer | generic package | Defines and provides operations required to manipulate a bounded FIFO buffer |
| Unbounded_FIFO_Buffer | generic package | Defines and provides operations required to manipulate an unbounded FIFO buffer |
| Nonblocking_Circular_Buffer | generic package | Defines and provides operations required to manipulate a nonblocking circular buffer |
| Unbounded_Priority_Queue | generic package | Defines and provides operations required to manipulate an unbounded priority queue |
| Bounded_Stack | generic package | Defines and provides operations required to manipulate a bounded stack |
| Unbounded_Stack | generic package | Defines and provides operations required to manipulate an unbounded stack |

3.6.9.1.9 PART DESIGN

3.6.9.1.9.1 BOUNDED_FIFO_BUFFER (PACKAGE) (CATALOG #P324-0)

This generic package defines the data type and contains the operations required to perform first-in-first-out buffering operations on incoming data. The head always points to a dummy node. The first node following the dummy node contains the next piece of data to be retrieved. The tail always points to where the next element should be added. If the tail points to the element immediately in front of the head, the buffer is empty. If the tail points to the same element as the head, the buffer is full. Since the buffer is implemented as an array, the head and tail will advance through the array in a circular fashion, but no overwriting of data currently in the buffer will be permitted.





This part has been designed so that the following routines may be used by two tasks of different priorities as long as one is only putting things in the buffer and the other is only removing things from the buffer:

- o Add_Element
- o Buffer_Status
- o Retrieve_Element
- o Peek

Neither Buffer_Length or Clear_Buffer should be used by tasks of differing priorities as described above. Buffer_Length should not be used since the internally stored buffer length could have become corrupted although the buffer itself remain intact. Clear_Buffer should be not be called since it could result in the buffer becoming corrupted.

The following table shows which exceptions are raised by which unit in this package:

| Name of routine \ Exception raising exception \ raised => | Buffer_Full | Buffer_Empty |
|---|---|---|
| Add_Element | X | |
| Retrieve_Element | | X |
| Peek | | X |

3.6.9.1.9.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP required R125.

3.6.9.1.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Elements | private | User defined type of data contained in the buffer |

Data objects:

The following table summarizes the generic formal objects required by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Initial_ Buffer_Size | POSITIVE | N/A | Maximum number of elements which can be in the buffer at any given time |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following table describes the exceptions exported by this part:

| Name | Description |
|---|---|
| Buffer_Empty | Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer |
| Buffer_Full | Error condition raised if an attempt is made to add elements to a full buffer |

Data types:

One of the data types exported by this part is "buffers". Since this is a limited private type, the only way the user can access the buffer is through functions and procedures contained in this part.

The following chart summarizes the data types exported by this part:

| Name | Type | Range | Description |
|---|---|---|---|
| Buffer_Range | NATURAL subtype | 0 .. Buffer_Size | Used to dimension the list of elements |
| Buffers | limited private | N/A | List of data along with relevant information |
| Buffer_ Statuses | discrete type | Empty, Available, Full | Used to indicate the status of the buffer |

Data objects:

The following table describes the data objects exported by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Buffer_Size | POSITIVE | Initial_ Buffer_Size | Number of usable elements in a buffer |

3.6.9.1.9.1.3 LOCAL ENTITIES

None.

3.6.9.1.9.1.4 INTERRUPTS

None.

3.6.9.1.9.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Abstract_Data_Structures;
...
   package ADS renames Abstract_Data_Structures;
   ...
   type Messages is ...
   ...
   Max_Message_Count : constant := 15;
   ...
   package Bounded_FIFO is new ADS.Bounded_FIFO_Buffer
             (Elements            => Messages,
              Initial_Buffer_Size => Max_Message_Count);
   ...
   Message_Buffer : Bounded_FIFO.Buffers;
   New_Message    : Messages;
   Next_Message   : Messages;
   ...
      begin
         ...
         Bounded_FIFO.Clear_Buffer(Message_Buffer);
         ...
         Bounded_FIFO.Add_Element(New_Message, Message_Buffer);
         ...
         Bounded_FIFO.Retrieve_Element(Message_Buffer, Next_Message);
         ...
```

3.6.9.1.9.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.9.1.9.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Clear_Buffer | procedure | Clears the contents of the buffer |
| Add_Element | procedure | Adds an element to the end of the buffer |
| Retrieve_Element | procedure | Removes an element from the front of the buffer |
| Peek | function | Looks at first element at the front of the buffer without altering the buffer's contents |
| Buffer_Status | function | Returns the status of the buffer |
| Buffer_Length | function | Returns the number of elements currently in the buffer |

3.6.9.1.9.1.8 PART DESIGN

None.

3.6.9.1.9.2 UNBOUNDED_FIFO_BUFFER (PACKAGE) (CATALOG #P325-0)

This generic package defines the data type and contains the operations required to perform first-in-first-out buffering operations on incoming data. The head of the buffer always points to a dummy node. The first node following the dummy node contains the next piece of data to be retrieved. The tail always points to the node containing the last element added to the buffer. If the tail points to the same node as the head, the buffer is empty.

A buffer must be initialized before it is used. If an attempt is made to use an uninitialized buffer, the exception Buffer_Not_Initialized will be raised. The Initialized_Buffer function returns an initialized buffer. The Clear_Buffer procedure returns the nodes of a buffer to the available space list and then returns an initialized buffer.

An available space list is maintained local to this part. When this part is elaborated the available space list will have a dummy node plus Initial_Available_Space_Size nodes. When nodes are added to the buffer, the Add_Element routine will try to get a node from the available space list before attempting to allocate more memory. When the Retrieve_Element routine is called, the unused node will be returned to the available space list for later use. The memory committed to the available space may be deallocated by calling the Free_Memory procedure.

The following table describes the exceptions raised by this part:

| Name | When/Why Raised |
|---|---|
| Storage_Error | Raised during elaboration of this package or by one of the following routines if an attempt is made to allocate more memory than is available:<br>o Initialized_Buffer<br>o Add_Element |
| Buffer_Empty | Raised by the following routines if an attempt is made to access an empty buffer:<br>o Peek<br>o Retrieve_Element |
| Buffer_Not_Initialized | Raised by the following routines if an attempt is made to use an uninitialized buffer:<br>o Retrieve_Element<br>o Add_Element<br>o Peek<br>o Buffer_Length<br>o Clear_Buffer |

3.6.9.1.9.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R164.

3.6.9.1.9.2.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Elements | private | User defined type of data contained in the buffer |

Data objects:

The following table summarizes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Available_Space_Size | NATURAL | Number of nodes to be initially placed in the available space list |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following table describes the exceptions exported by this part:

| Name | Description |
|---|---|
| Buffer_Empty | Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer |
| Buffer_Not_Initialized | Raised if an attempt is made to use an uninitialized buffer |
| Storage_Error | Raised if an attempt is made to allocate more memory than is available |

Data types:

The data type exported by this part is "buffers". Since this is a limited private type, the only way the user can access the buffer is through functions and procedures contained in this part.

The following chart summarizes the data types exported by this part:

| Name | Type | Range | Description |
|---|---|---|---|
| Buffers | limited private | N/A | List of data along with relevant information |
| Buffer_Statuses | discrete type | Uninitialized, Empty, Available | Used to indicate the status of the buffer |

3.6.9.1.9.2.3 LOCAL ENTITIES

Data structures:

An available space list is maintained local to this part's package body.

3.6.9.1.9.2.4 INTERRUPTS

None.

3.6.9.1.9.2.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Abstract_Data_Structures;
...
   package ADS renames Abstract_Data_Structures;
   type Messages is ...
   ...
   Initial_Space_Size : constant := 15;
   ...
```

```
package Unbounded_FIFO is new ADS.Unbounded_FIFO_Buffer
          (Elements                    => Messages,
           Initial_Available_Space_Size => Initial_Space_Size);
...
Message_Buffer : Unbounded_FIFO.Buffers;
New_Message    : Messages;
Next_Message   : Messages;
...
   begin
      ...
      Unbounded_FIFO.Initialize_Buffer(Message_Buffer);
      ...
      Unbounded_FIFO.Add_Element(New_Message, Message_Buffer);
      ...
      Unbounded_FIFO.Retrieve_Element(Message_Buffer, Next_Message);
      ...
```

3.6.9.1.9.2.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.9.1.9.2.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Initialize_ Buffer | function | Returns an initialized buffer |
| Clear_Buffer | procedure | Clears the contents of the buffer |
| Free_Memory | procedure | Deallocates the memory allocated to the available space list |
| Add_Element | procedure | Adds an element to the end of the buffer |
| Retrieve_ Element | procedure | Removes an element from the front of the buffer |
| Peek | function | Looks at first element at the front of the buffer without altering the buffer's contents |
| Buffer_Status | function | Returns the status of the buffer |
| Buffer_Length | function | Returns the number of elements currently in the buffer |

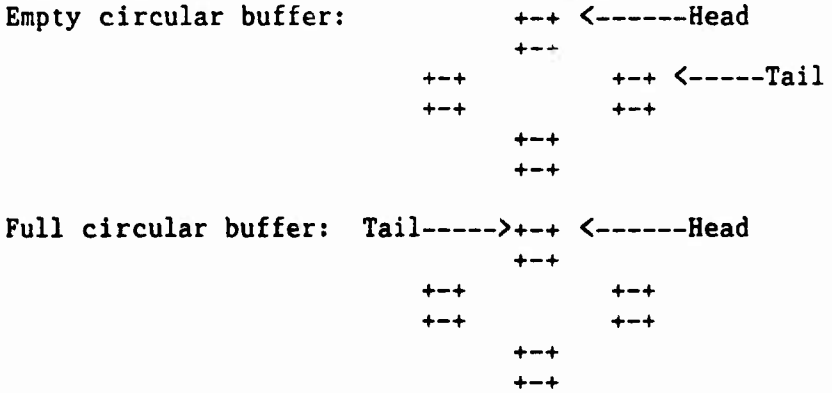
3.6.9.1.9.2.8 PART DESIGN

None.

3.6.9.1.9.3 NONBLOCKING_CIRCULAR_BUFFER (PACKAGE) (CATALOG #P326-0)

This generic package defines the data type and contains the operations required to perform circular buffering operations on incoming data. These operations are performed in a non-blocking fashion such that if the buffer is full,

incoming data will overwrite old data. The head of the buffer always points to a dummy node. The first node following the dummy node contains the next piece of data to be retrieved. The tail always points to where the next element should be added. If the tail points to the element immediately in front of the head, the buffer is empty. If the tail points to the same element as the head, the buffer is full. This is illustrated below.




The following table shows which exceptions are raised by which unit in this package:

| Name of routine raising exception \ Exception raised => | Buffer_Empty |
|---|---|
| Retrieve_Element | X |
| Peek | X |

3.6.9.1.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R126.

3.6.9.1.9.3.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Elements | private | User defined type of data contained in the buffer |

Data objects:

The following table summarizes the generic formal objects required by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Initial_Buffer_Size | POSITIVE | N/A | Maximum number of elements which can be in the buffer at any given time |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following table describes the exceptions exported by this part:

| Name | Description |
|---|---|
| Buffer_Empty | Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer |

Data types:

One of the data types exported by this part is "buffers". Since this is a limited private type, the only way the user can access the buffer is through functions and procedures contained in this part.

The following chart summarizes the data types exported by this part:

| Name | Type | Range | Description |
|---|---|---|---|
| Buffer_Range | NATURAL subtype | 0 .. Buffer_Size | Used to dimension the list of elements |
| Buffers | limited private | N/A | List of data along with relevant information |
| Buffer_Statuses | discrete type | Empty, Available, Full | Used to indicate the status of the buffer |

Data objects:

The following table describes the data objects exported by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Buffer_Size | POSITIVE | Initial_Buffer_Size | Number of usable elements in a buffer |

3.6.9.1.9.3.3 LOCAL ENTITIES

None.

3.6.9.1.9.3.4 INTERRUPTS

None.

3.6.9.1.9.3.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Abstract_Data_Structures;
...
   package ADS renames Abstract_Data_Structures;
   ...
   type Messages is ...
   ...
   Max_Message_Count : constant := 15;
   ...
   package Circ_Buffer is new ADS.Nonblocking_Circular_Buffer
             (Elements            => Messages,
              Initial_Buffer_Size => Max_Message_Count);
   ...
   Message_Buffer : Circ_Buffer.Buffers;
   New_Message    : Messages;
   Next_Message   : Messages;
   ...
      begin
         ...
         Circ_Buffer.Clear_Buffer(Message_Buffer);
         ...
         Circ_Buffer.Add_Element(New_Message, Message_Buffer);
         ...
         Circ_Buffer.Retrieve_Element(Message_Buffer, Next_Message);
         ...
```

3.6.9.1.9.3.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.9.1.9.3.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Clear_Buffer | procedure | Clears the contents of the buffer |
| Add_Element | procedure | Adds an element to the end of the buffer |
| Retrieve_Element | procedure | Removes an element from the from of the buffer |
| Peek | function | Looks at first element at the front of the buffer without altering the buffer's contents |
| Buffer_Status | function | Indicates the status of the buffer |
| Buffer_Length | function | Returns the number of elements currently in the buffer |

3.6.9.1.9.3.8 PART DESIGN

None.

3.6.9.1.9.4 UNBOUNDED_PRIORITY_QUEUE (CATALOG #P327-0)

This generic package defines the data type and contains the operations required to perform priority queueing operations on incoming data. The head of the queue always points to a dummy node. The node following the dummy node contains the element with the highest priority. The tail always points to the element with the lowest priority.

The elements will be ordered in the queue such that:

1) Elements with higher priorities are placed before those with lower priorities.
2) Elements with the same priority are arrange in the queue in a first-in-first-out manner.

A queue must be initialized before it is used. If an attempt is made to use an uninitialized queue, the exception Queue_Not_Initialized will be raised. The Initialized_Queue function returns an initialized queue. The Clear_Queue procedure returns the nodes of a queue to the available space list and then returns an initialized queue.

An available space list is maintained local to this part. When this part is elaborated the available space list will have a dummy node plus Initial_-Available_Space_Size nodes. When nodes are added to the queue, the Add_Element routine will try to get a node from the available space list before attempting to allocate more memory. When the Retrieve_Element routine is called, the unused node will be returned to the available space list for later use. The memory committed to the available space may be deallocated by calling the Free_Memory procedure.

The following table describes the exceptions raised by this part:

| Name | When/Why Raised |
|---|---|
| Storage_Error | Raised during elaboration of this package or by one of the following routines if an attempt is made to allocate memory when no more is available:<br>o Initialized_Queue<br>o Add_Element |
| Queue_Empty | Raised by the following routines if an attempt is made to access an empty queue:<br>o Retrieve_Element<br>o Peek |
| Queue_Not_Initialized | Raised by the following routines if an attempt is made to manipulate an uninitialized queue:<br>o Add_Element<br>o Retrieve_Element<br>o Queue_Length<br>o Peek<br>o Clear_Queue |

#### 3.6.9.1.9.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R165.

#### 3.6.9.1.9.4.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Elements | private | User defined type of data contained in the queue |
| Priorities | private | User defined type determining priority of the node |

Data objects:

The following table summarizes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Available_Space_Size | NATURAL | Number of available nodes to be initially placed in the available space list |

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| ">" | function | Used to determine ordering of priorities |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following table describes the exceptions exported by this part:

| Name | Description |
|---|---|
| Queue_Empty | Error condition raised if an attempt is made to look at or retrieve elements from an empty queue |
| Queue_Not_Initialized | Indicates an attempt was made to use an uninitialized queue |
| Storage_Error | Raised if an attempt is made to allocate more memory than is available |

Data types:

The data type exported by this part is "queues". This type consists of the pointers to the nodes of user-defined elements and priorities, along with pertinent information about the queue. Since it is a limited private type, the only way the user can gain access to the queue is through functions and procedures contained in this part.

The following chart summarizes the data types exported by this part:

| Name | Type | Range | Description |
|---|---|---|---|
| Queues | limited private | N/A | List of data along with relevant information |
| Queue_Statuses | discrete type | Uninitialized, Empty, Available | Used to indicate the status of the queue |

3.6.9.1.9.4.3 LOCAL ENTITIES

Data structures:

An available space list is maintain local to this part's package body.

3.6.9.1.9.4.4 INTERRUPTS

None.

3.6.9.1.9.4.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Abstract_Data_Structures;
...
   package ADS renames Abstract_Data_Structures;
   ...
   type Messages is ...
   ...
   Initial_Space_Size : constant := 15;
   ...
   package Unbounded_Priority is new ADS.Unbounded_Priority_Queue
             (Elements                    => Messages,
              Initial_Available_Space_Size => Initial_Space_Size);
   ...
   Message_Queue : Unbounded_Priority.Queues;
   New_Message   : Messages;
   Next_Message  : Messages;
   ...
      begin
         ...
         Unbounded_Priority.Initialize(Queue);
         ...
         Unbounded_Priority.Add_Element(New_Message, Message_Queue);
         ...
         Unbounded_Priority.Retrieve_Element(Message_Queue, Next_Message);
         ...
```

3.6.9.1.9.4.6 GLOBAL PROCESSING

There is no global processing performed by this LCSC.

3.6.9.1.9.4.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Initialize | function | Returns an initialized priority queue |
| Clear_Queue | procedure | Clears the contents of the queue |
| Free_Memory | procedure | Deallocates the memory allocation to the available space list |
| Add_Element | procedure | Adds an element to the input side of the queue |
| Retrieve_Element | procedure | Removes an element from the output side of the queue |
| Peek | function | Looks at first element on the front of the queue without altering the queue's contents |
| Queue_Status | function | Returns the status of the queue |
| Queue_Length | function | Returns the number of elements currently in the queue |

3.6.9.1.9.4.8 PART DESIGN

None.

3.6.9.1.9.5 BOUNDED_STACK (PACKAGE) (CATALOG #P328-0)

This generic package defines the data type and contains the operations required to perform last-in-first-out stacking operations on incoming data. The top of the stack always points to the last element added to the stack and the next element to be removed. When top equals 0, the stack is empty. When it equals Stack_Size, the stack is full.

The following table shows which exceptions are raised by which unit in this package:

| Name of routine \ Exception raising exception \ raised => | Stack_Full | Stack_Empty |
|---|---|---|
| Add_Element | X | |
| Retrieve_Element | | X |
| Peek | | X |

3.6.9.1.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R166.

3.6.9.1.9.5.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Elements | private | User defined type of data contained in the stack |

Data objects:

The following table summarizes the generic formal objects required by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Initial_ Stack_Size | POSITIVE | N/A | Maximum number of elements which can be in the stack at any given time |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following table describes the exceptions exported by this part:

| Name | Description |
|---|---|
| Stack_Empty | Error condition raised if an attempt is made to look at or retrieve elements from an empty stack |
| Stack_Full | Error condition raised if an attempt is made to add elements to a full stack |

Data types:

One of the data types exported by this part is "stacks". Since this is a limited private type, the only way the user can access the stack is through functions and procedures contained in this part.

The following chart summarizes the data types exported by this part:

| Name | Type | Range | Description |
|---|---|---|---|
| Stack_ Length_ Range | POSITIVE subtype | 1 .. Stack_Size | Used to dimension the list of elements |
| Stacks | limited private | N/A | List of data along with relevant information |
| Stack_ Statuses | discrete type | Empty, Available, Full | Used to indicate the status of the stack |

Data objects:

The following table describes the data objects exported by this part:

| Name | Type | Value | Description |
|---|---|---|---|
| Stack_Size | POSITIVE | Initial_Stack_Size | Number of elements in the stack |

3.6.9.1.9.5.3 LOCAL ENTITIES

None.

3.6.9.1.9.5.4 INTERRUPTS

None.

3.6.9.1.9.5.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Abstract_Data_Structures;
...
   package ADS renames Abstract_Data_Structures;
   ...
   type Messages is ...
   ...
   Max_Message_Count : constant := 15;
   ...
   package Bnded_Stack is new ADS.Bounded_Stack
             (Elements           => Messages,
              Initial_Stack_Size => Max_Message_Count);
   package Bounded_Stack renames Bnded_Stack;
   ...
   Message_Stack : Bounded_Stack.Buffers;
   New_Message   : Messages;
   Next_Message  : Messages;
   ...
      begin
         ...
         Bounded_Stack.Clear_Stack(Message_Stack);
         ...
         Bounded_Stack.Add_Element(New_Message, Message_Stack);
         ...
         Bounded_Stack.Retrieve_Element(Message_Stack, Next_Message);
         ...
```

3.6.9.1.9.5.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.9.1.9.5.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Clear_Stack | procedure | Returns a cleared stack |
| Add_Element | procedure | Adds an element to the top of the stack |
| Retrieve_Element | procedure | Removes an element from the top of the stack |
| Peek | function | Looks at first element on top of the stack without altering the stack's contents |
| Stack_Status | function | Returns the current status of the stack |
| Stack_Length | function | Returns the number of elements currently in the stack |

3.6.9.1.9.5.8 PART DESIGN

None.

3.6.9.1.9.6 UNBOUNDED_STACK (CATALOG #P329-0)

This generic package performs last-in-first-out stacking operations on incoming data. The head of the stack always points to the last element added to the stack and the next element to be removed. The tail always points to a dummy node located below the oldest element on the stack. If head and tail point to the same node, the stack is empty.

An available space list is maintained local to this part. When this part is elaborated the available space list will have a dummy node plus Initial_Available_Space_Size nodes. When nodes are added to the buffer, the Add_Element routine will try to get a node from the available space list before attempting to allocate more memory. When the Retrieve_Element routine is called, the unused node will be returned to the available space list for later use. The memory committed to the available space may be deallocated by calling the Free_Memory procedure.

The following table describes the exceptions raised by this part:

| Name | When/Why Raised |
|---|---|
| Storage_Error | Raised during elaboration of this package or by one of the following routines if an attempt is made to allocate memory when no more is available:<br>o Initialized_Stack<br>o Add_Element |
| Stack_Empty | Raised by the following routines if an attempt is made to access an empty stack:<br>o Peek<br>o Retrieve_Element |
| Stack_Not_Initialized | Raised by the following routines if an attempt is made to use an uninitialized stack:<br>o Clear_Buffer<br>o Retrieve_Element<br>o Add_Element<br>o Peek<br>o Buffer_Length |

3.6.9.1.9.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R167.

3.6.9.1.9.6.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Elements | private | User defined type of data contained in the stack |

Data objects:

The following table summarizes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Available_Space_Size | NATURAL | Number of nodes to be initially placed in the available space list |

EXPORTED EXCEPTIONS/TYPES/OBJECTS:

Exceptions:

The following table describes the exceptions exported by this part:

| Name | Description |
|---|---|
| Stack_Empty | Error condition raised if an attempt is made to look at or retrieve elements from an empty stack |
| Stack_Not_Initialized | Raised if an attempt is made to use an uninitialized stack |
| Storage_Error | Raised if an attempt is made to allocate more memory than is available |

Data types:

The data type exported by this part is "stacks". Since it is a limited private type, the only way the user can access the stack is through functions and procedures contained in this part.

The following chart summarizes the data types exported by this part:

| Name | Type | Range | Description |
|---|---|---|---|
| Stacks | limited private | N/A | List of data along with relevant information |
| Stack_Statuses | discrete type | Uninitialized, Empty, Available | Indicates the current status of the stack |

3.6.9.1.9.6.3 LOCAL ENTITIES

Data structures:

This part maintains an available space list local to the package body.

3.6.9.1.9.6.4 INTERRUPTS

None.

3.6.9.1.9.6.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with Abstract_Data_Structures;
...
   package ADS renames Abstract_Data_Structures;
   ...
   type Messages is ...
   ...
   Initial_Space_Size : constant := 15;
```

```
...
package Unbounded_Stck is new ADS.Unbounded_Stack
          (Elements                    => Messages,
           Initial_Available_Space_Size => Initial_Space_Size);
...
Message_Stack : Unbounded_Stck.Stacks;
New_Message   : Messages;
Next_Message  : Messages;
...
   begin
      ...
      Unbounded_Stck.Initialize(Message_Stack);
      ...
      Unbounded_Stck.Add_Element(New_Message, Message_Stack);
      ...
      Unbounded_Stck.Retrieve_Element(Message_Stack, Next_Message);
      ...
```

3.6.9.1.9.6.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.9.1.9.6.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Initialize | function | Returns an initialized stack |
| Clear_Stack | procedure | Clears the contents of the stack |
| Free_Memory | procedure | Deallocates the memory allocated to the available space list |
| Add_Element | procedure | Adds an element to the top of the stack |
| Retrieve_Element | procedure | Removes an element from the top of the stack |
| Peek | function | Looks at first element on top of the stack without altering the stack's contents |
| Stack_Status | function | Returns the status of the stack |
| Stack_Length | function | Returns the number of elements currently in the stack |

3.6.9.1.9.6.8 PART DESIGN

None.

(This page left intentionally blank.)

```
package Abstract_Data_Structures is

pragma PAGE;
   generic
     type Elements       is private;
     Initial_Buffer_Size : in POSITIVE;
   package Bounded_Fifo_Buffer is

--      --declarations

        Buffer_Full  : exception;
        Buffer_Empty : exception;

        Buffer_Size  : constant POSITIVE := Initial_Buffer_Size;

        subtype Buffer_Range is NATURAL range 0 .. Buffer_Size;
        type    Buffers      is limited private;

        type    Buffer_Statuses is (Empty, Available, Full);

--      --subroutine specifications

        procedure Clear_Buffer (Buffer : out Buffers);

        procedure Add_Element (New_Element : in     Elements;
                               Buffer      : in out Buffers);

        procedure Retrieve_Element (Buffer      : in out Buffers;
                                    Old_Element :    out Elements);

        function Peek (Buffer : in Buffers) return Elements;

        function Buffer_Status (Buffer : in Buffers) return Buffer_Statuses;

        function Buffer_Length (Buffer : in Buffers) return Buffer_Range;

--      --private section

        private

          type Lists   is array (Buffer_Range) of Elements;

          type Buffers is
            record
              Buffer_Length   : INTEGER      := 0;
              Head            : Buffer_Range := 0;
              Tail            : Buffer_Range := 1;
              LIST            : Lists;
            end record;

   end Bounded_Fifo_Buffer;

pragma PAGE;
   generic
     type Elements                 is private;
     Initial_Available_Space_Size : in NATURAL := 0;
   package Unbounded_Fifo_Buffer is
```

```
--      --declarations

        Buffer_Empty           : exception;
        Buffer_Not_Initialized : exception;
        STORAGE_ERROR          : exception renames STANDARD.STORAGE_ERROR;

        type Buffers is limited private;

        type Buffer_Statuses is (Uninitialized, Empty, Available);

--      --subroutine specifications

        procedure Initialize_Buffer (Buffer : in out Buffers);

        procedure Clear_Buffer (Buffer : in out Buffers);

        procedure Free_Memory;

        procedure Add_Element (New_Element : in     Elements;
                               Buffer      : in out Buffers);

        procedure Retrieve_Element (Buffer      : in out Buffers;
                                    Old_Element :    out Elements);

        function Peek (Buffer : in  Buffers) return Elements;

        function Buffer_Status (Buffer : in Buffers) return Buffer_Statuses;

        function Buffer_Length (Buffer : in Buffers) return NATURAL;

--      --private section

        private

          type Nodes;

          type Pointers is access Nodes;

          type Nodes is
            record
              Data : Elements;
              Next : Pointers := null;
            end record;

          type Buffers is
            record
              Current_Length : INTEGER  := -1;
              Head           : Pointers := null;
              Tail           : Pointers := null;
            end record;

    end Unbounded_Fifo_Buffer;

pragma PAGE;
    generic
      type Elements       is private;
```

```
    Initial_Buffer_Size : in POSITIVE;
  package Nonblocking_Circular_Buffer is

--    --declarations

    Buffer_Empty : exception;

    Buffer_Size  : constant POSITIVE := Initial_Buffer_Size;

    subtype Buffer_Range is NATURAL range 0 .. Buffer_Size;
    type    Buffers      is limited private;

    type    Buffer_Statuses is (Empty, Available, Full);

--    --subroutine specifications

    procedure Clear_Buffer (Buffer : out Buffers);

    procedure Add_Element (New_Element : in     Elements;
                           Buffer      : in out Buffers);

    procedure Retrieve_Element (Buffer      : in out Buffers;
                                Old_Element :    out Elements);

    function Peek (Buffer : in Buffers) return Elements;

    function Buffer_Status (Buffer : in Buffers) return Buffer_Statuses;

    function Buffer_Length (Buffer : in Buffers) return Buffer_Range;

--    --private section

    private

      type Lists   is array (Buffer_Range) of Elements;

      type Buffers is
        record
          Head           : Buffer_Range := 0;
          Tail           : Buffer_Range := 1;
          Current_Length : Buffer_Range := 0;
          LIST           : Lists;
        end record;

  end Nonblocking_Circular_Buffer;

pragma PAGE;
  generic
    type Elements   is private;
    type Priorities is private;
    Initial_Available_Space_Size : in NATURAL := 0;
    with function ">" (Left  : in Priorities;
                       Right : in Priorities) return BOOLEAN is <>;
  package Unbounded_Priority_Queue is

--    --declarations
```

```
      Queue_Empty           : exception;
      Queue_Not_Initialized : exception;

      type Queues is limited private;

      type Queue_Statuses is (Uninitialized, Empty, Available);

--    --subroutine specifications

      procedure Initialize (Queue : in out Queues);

      procedure Clear_Queue (Queue : in out Queues);

      procedure Free_Memory;

      procedure Add_Element (New_Element  : in     Elements;
                             New_Priority : in     Priorities;
                             Queue        : in out Queues);

      procedure Retrieve_Element (Queue       : in out Queues;
                                  Old_Element :    out Elements);

      function Peek (Queue : in Queues) return Elements;

      function Queue_Status (Queue : in Queues) return Queue_Statuses;

      function Queue_Length (Queue : in Queues) return NATURAL;

--    --private section

      private

        type Nodes;

        type Pointers is access Nodes;

        type Nodes is
          record
            PRIORITY : Priorities;
            Data     : Elements;
            Next     : Pointers := null;
          end record;

        type Queues is
          record
            Current_Length : INTEGER  := -1;
            Head           : Pointers := null;
            Tail           : Pointers := null;
          end record;

   end Unbounded_Priority_Queue;

pragma PAGE;
   generic
     type Elements      is private;
     Initial_Stack_Size : in POSITIVE;
   package Bounded_Stack is
```

```
--      --declarations

        Stack_Full  : exception;
        Stack_Empty : exception;

        Stack_Size  : constant POSITIVE := Initial_Stack_Size;

        subtype Stack_Length_Range is NATURAL range 0 .. Stack_Size;
        type    Stacks             is limited private;

        type Stack_Statuses is (Empty, Available, Full);

--      --subroutine specifications

        procedure Clear_Stack (Stack : out Stacks);

        procedure Add_Element (New_Element : in     Elements;
                               Stack       : in out Stacks);

        procedure Retrieve_Element (Stack       : in out Stacks;
                                    Old_Element :    out Elements);

        function Peek (Stack : in Stacks) return Elements;

        function Stack_Status (Stack : in Stacks) return Stack_Statuses;

        function Stack_Length (Stack : in Stacks) return Stack_Length_Range;

--      --private section

        private

           subtype Stack_Dimensions is Stack_Length_Range
                                  range 1 .. Stack_Length_Range'LAST;

           type Lists is array (Stack_Dimensions) of Elements;

           type Stacks is
              record
                 Top            : Stack_Length_Range := 0;
                 LIST           : Lists;
              end record;

     end Bounded_Stack;

  pragma PAGE;
     generic
       type Elements is private;
       Initial_Available_Space_Size : in NATURAL := 0;
     package Unbounded_Stack is

--      --declarations

        Stack_Empty           : exception;
        Stack_Not_Initialized : exception;
        STORAGE_ERROR         : exception renames STANDARD.STORAGE_ERROR;
```

```
      type Stacks is limited private;

      type Stack_Statuses is (Uninitialized, Empty, Available);

--    --subroutine specifications

      procedure Initialize (Stack : in out Stacks);

      procedure Clear_Stack (Stack : in out Stacks);

      procedure Free_Memory;

      procedure Add_Element (New_Element : in     Elements;
                             Stack       : in out Stacks);

      procedure Retrieve_Element (Stack       : in out Stacks;
                                  Old_Element :    out Elements);

      function Peek (Stack : in Stacks) return Elements;

      function Stack_Status (Stack : in Stacks) return Stack_Statuses;

      function Stack_Length (Stack : in Stacks) return NATURAL;

--    --private section

      private

        type Nodes;

        type Pointers is access Nodes;

        type Nodes is
          record
            Data : Elements;
            Next : Pointers := null;
          end record;

        type Stacks is
          record
            Current_Length : INTEGER  := -1;
            Top            : Pointers := null;
            Bottom         : Pointers := null;
          end record;

   end Unbounded_Stack;

end Abstract_Data_Structures;
```

### 3.6.10 GENERAL UTILITIES

(This page left intentionally blank.)

3.6.10.1 GENERAL_UTILITIES TLCSC P361 (CATALOG P265-0)

This package provides a group of general utility routines used in a missile system.

3.6.10.1.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Instruction_Set_Test | R141 |

3.6.10.1.2 INPUT/OUTPUT

None.

3.6.10.1.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.10.1.4 LOCAL ENTITIES

None.

3.6.10.1.5 INTERRUPTS

None.

3.6.10.1.6 TIMING AND SEQUENCING

None.

3.6.10.1.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.10.1.8 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Instruction_Set_Test | generic function | checks for proper processor operation |

3.6.10.1.9 PART DESIGN

3.6.10.1.9.1 INSTRUCTION_SET_TEST (CATALOG P266-0)

This part is a generic function which checks for proper processor operation by executing a function and comparing the result to the expected result. If the expected and derived values match, "True" is returned. The part's generic parameter may be any type, but a Test function must be supplied which matches the parameter defined in the specification.

3.6.10.1.9.1.1 REQUIREMENTS ALLOCATION

This part meets requirement R141.

3.6.10.1.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Return_Values | private | may be any type. |

Subprograms:

The following table describes the generic formal subroutines required by this part:

| Name | Type | Description |
|---|---|---|
| Test | function | the function to be tested, it must return a value of Return_Values type. |

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

| Name | Type | Mode | Description |
|---|---|---|---|
| Correct_Answer | Return_Values | in | The correct return value for the function. |

3.6.10.1.9.1.3 INTERRUPTS

None.

3.6.10.1.9.1.4 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
with General_Utilties;
procedure Sample:

   Expected_Result : Float;
   Test_Result     : BOOLEAN;

   function My_Test return Float is
   begin
      return 1.0;
   end My_Test;

   Test_It is new General_Utilities.Instruction_Set_Test
                                      ( Return_Values => Float,
                                        Test          => My_Test );

    begin
       Expected_Result := 1.0;
       Test_Result := Test_It( Expected_Result );
    end Sample;
```

3.6.10.1.9.1.5 GLOBAL PROCESSING

There is no global processing performed by this Unit.

3.6.10.1.9.1.6 DECOMPOSITION

None.

(This page left intentionally blank.)

```
package General_Utilities is

   generic
      type Return_Values is private;
      with function Test return Return_Values is <>;
   function Instruction_Set_Test (Correct_Answer : Return_Values)
                                                        return BOOLEAN;

end General_Utilities;
```

(This page intentionally left blank.)

3.6.10.2 COMMUNICATION_PARTS TLCSC P602 (CATALOG #P689-0)

This package provides a group of communication routines used in a missile system.

3.6.10.2.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Update_Exclusion | R137 |

3.6.10.2.2 INPUT/OUTPUT

None.

3.6.10.2.3 UTILIZATION OF OTHER ELEMENTS

None.

3.6.10.2.4 LOCAL ENTITIES

None.

3.6.10.2.5 INTERRUPTS

None.

3.6.10.2.6 TIMING AND SEQUENCING

None.

3.6.10.2.7 GLOBAL PROCESSING

There is no global processing performed by this TLCSC.

3.6.10.2.8 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Update_Exclusion | generic package | provides a mechanism for insuring that data accessed by more than one asynchronous task (with priorities supported) is properly protected for such accesses. |

3.6.10.2.9 PART DESIGN

3.6.10.2.9.1 UPDATE_EXCLUSION (CATALOG #P690-0)

This part is a generic package containing a task providing a mechanism for insuring that data accessed by more than one asynchronous task is properly protected for such accesses. The part's generic parameter can be any type.

3.6.10.2.9.1.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of requirements to this part:

| Name | Requirements Allocation |
|---|---|
| Update_Exclusion | R137 |

3.6.10.2.9.1.2 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

| Name | Type | Description |
|---|---|---|
| Element_Type | private | Allows any type to be protected |

Data objects:

The following table describes the generic formal objects required by this part:

| Name | Type | Description |
|---|---|---|
| Initial_Value | Element_Type | Allows the data type to be initialized so that the first time Start_Update_Request is called a constraint error is not raised by some uninitialized value. |

3.6.10.2.9.1.3 LOCAL ENTITIES

None.

3.6.10.2.9.1.4 INTERRUPTS

None.

3.6.10.2.9.1.5 TIMING AND SEQUENCING

The following shows a sample usage of this part:

```
With Communication_Parts;
With STANDARD;

procedure Sample is
   type State   is ( Startup, Running, Locked, Waiting );
   Begin_State   : State := Startup;
   Current_State : State;
   Updated_State : State;

   package Change_State is new Communication_Parts.Update_Exclusion
                                      ( Element_Type  => State,
                                        Initial_Value => Begin_State );
   Result   : Change_State.Rendezvous_Flags;
   My_Id    : Change_State.Rendezvous_Ids;
   Time_Now : STANDARD.DURATION;

   begin
      Change_State.Attempt_Start_Update( Current_State,
                                         My_Id,
                                         Result );
      -- Value of Current_State is 'Startup'
      -- The object is locked and cannot be read or written here
      -- Note that attempted rendezvous will not be acknowledged
      -- if made here.
      Current_State := Running;
      Change_State.Attempt_Complete_Update( Current_State,
                                            My_Id,
                                            Result );
      -- Value of Current_State is 'Running'
      -- The object is available for reads or updates here
      Change_State.Attempt_Read( Updated_State, );
      -- Value of Updated_State is 'Running'
   end Sample;
```

3.6.10.2.9.1.6 GLOBAL PROCESSING

There is no global processing performed by this LLCSC.

3.6.10.2.9.1.7 DECOMPOSITION

The following table describes the decomposition of this part:

| Name | Type | Description |
|---|---|---|
| Attempt_Read | procedure | Attempts to read the protected data. If unable, does not wait. |
| Attempt_Read_Wait | procedure | Attempts to read the protected data. If unable, waits indefinitely until it can. |
| Attempt_Read_Delay | procedure | Attempts to read the protected data, If unable, waits the specified time |
| Attempt_Start_Update | procedure | Attempts to start an update. If unable, does not wait. |
| Attempt_Start_Update_Wait | procedure | Attempts to start an update. If unable, waits indefinitely until it can. |
| Attempt_Start_Update_Delay | procedure | Attempts to start an update. If unable, delays the specified amount. |
| Attempt_Complete_Update | procedure | Attempts to complete an update. If unable does not wait. |

3.6.10.2.9.1.8 PART DESIGN

None.

```
package Communication_Parts is

   generic
      type Element_Type is private;
      Initial_Value : in Element_Type;
   package Update_Exclusion is

      type Rendezvous_Flags is ( Success, Failure, Bad_Id );
      type Rendezvous_Ids is range 0..1000;

      Id   : Rendezvous_Ids := 1;

      procedure Attempt_Read( Requested_Data : in out Element_Type;
                              Result         : out Rendezvous_Flags );

      procedure Attempt_Read_Wait( Requested_Data : in out Element_Type;
                                   Result         : out Rendezvous_Flags );

      procedure Attempt_Read_Delay( Requested_Data : in out Element_Type;
                                    Result         : out Rendezvous_Flags;
                                    Delay_Time     : in  DURATION );

      procedure Attempt_Start_Update( Old_Data : in out Element_Type;
                                      New_Id   : out Rendezvous_Ids;
                                      Result   : out Rendezvous_Flags );

      procedure Attempt_Start_Update_Wait( Old_Data : in out Element_Type;
                                           New_Id   : out Rendezvous_Ids;
                                           Result   : out Rendezvous_Flags);

      procedure Attempt_Start_Update_Delay( Old_Data : in out Element_Type;
                                            New_Id   : out Rendezvous_Ids;
                                            Result   : out Rendezvous_Flags;
                                            Time     : in  DURATION );

      procedure Attempt_Complete_Update( New_Data  : in  Element_Type;
                                         Passed_Id : in  Rendezvous_Ids;
                                         Result    : out Rendezvous_Flags );

   end Update_Exclusion;

end Communication_Parts;
```

(This page intentionally left blank.)

4 NOT USED

(This page intentionally left blank.)

5 NOT USED

(This page intentionally left blank.)

## 6 NOTES

This paragraph does not apply to this TLDD.

(This page intentionally left blank.)

APPENDIX I

MODIFICATIONS TO DI-MCCR-80012

## 10.1 REQUIREMENTS FOR DOCUMENTING DESIGN OF REUSABLE PARTS

### 10.1.1 PROBLEMS IN USING DI-MCCR-80012

The documentation of the top-level design for the CAMP parts must describe the architecture of part packages and detail the interfaces between packages. This will require TLCSCs which address the following issues:

- o The package context (the list of external packages which are needed)
- o The decomposition of the TLCSC in to LLCSCs
- o Ada design of the specification of the TLCSC and its LLCSCs
- o Major entities which are local to the package body
- o Externally callable entries (where tasking is used)
- o Requirements for instantiation and other use of a part
- o Global processing and output

These requirements must be met both in the TLDD and in the header of the design code itself.

The Data Item Descriptor for the Software Top-Level Design Document (DI-MCCR-80012) does not adequately cover these issues. The DID seems to be directed towards a design which features data passing through shared data, rather than parameter passing, and parameterless subroutines employed for structural reasons, rather than functional or object-oriented decomposition. This architecture for a TLCSC is not compatible with the object-oriented nature of an Ada package specification. Therefore, the TLDD is not sufficient for our documentation needs.

Much of the information that properly belongs to a TLCSC designed using Ada has been placed in the Software Detailed Design Document (e.g., the TLCSC decomposition, and LLCSC interfacing). The CAMP project has determined that this information must appear in the top-level design description. This will require that the DID for top-level design be modified to include architectural information highlighting the structure of the TLCSC down to the unit level, where units are externally callable. It should also include structural information which is required for the detailed design of these external interfaces. In Ada terms, the TLDD will document the Ada specification plus major data structures and processing needs of the package body.

The Detailed Design Document will describe the implementation of all of the top-level design requirements, for both the bundled version of parts and the unbundled version. The DDD must contain the full package body for all

TLCSCs plus those source code segments which are used to build the Ada design code. The DDD will include the design code for individual parts, the CAMP library structure and the CAMP source text structure.

### 10.1.2 DESIGN CODE HEADER INFORMATION FOR TOP-LEVEL DESIGN

```
--TLCSC Name
--  The name shall be descriptive of the processing performed by the TLCSC.

--TLCSC Identification Number
--  The design identification number used to identify the TLCSC for
--  configuration management.

--Detailed overview of TLCSC purpose
--  For generic units, this section shall also provide detailes of the
--  capabilities provided by generic parameters (analogous to states of
--  operation)

--Requirements trace
--  Document SRS requirements met by the TLCSC.
--  May reference a block diagram to illustrate source of inputs and
--  destination of outputs of TLCSC.  Diagram should allow allocation of
--  CSCI requirements.

--Context of TLCSC
--  Describe context of TLCSC (packages which are with'd, or are otherwise
--  visible and are referenced in the TLCSC).  Describe what services of
--  these packages (data types, objects, functions) are used.  This will
--  describe global data used by the TLCSC.

--Exported Entities
--  Describe data objects, data types, subprograms, and packages defined by
--  the TLCSC.  Summarize in tabular form to show services exported by the
--  TLCSC.  Also, describe in detail all exported entities:

--  Data objects
--    Describe data objects exported by the TLCSC.  This shall include:
--      o Name of object
--      o Type of data
--      o Value, if a constant
--      o Brief description of data

--  Data types
--    Describe data types exported by the TLCSC.  This shall include:
--      o Name of type
--      o Range of type
--      o Predefined operators
--      o Special operators
--      o Brief description of type

--  Subprograms
--    Describe the deomposition of the TLCSC into processing entities which
--    shall become lower level CSCs and units.  For each LLCSC or unit
--    defined by the decomposition, provide the following information:
--      o Name
```

```
--       o Abstract describing purpose of subprogram.  For generic subprograms
--         this shall include detailes of the capabilities provided by
--         generic parameters
--       o Requirements trace
--       o Input data (parameters or global data)
--       o Processing algorithms
--       o Error conditions not handled immediately by the entity
--       o Outputs (parameters or global data)

--  Packages
--    Describe the decomposition of the TLCSC into packages which shall
--    become lower level CSCs and units.  For each package defined by this
--    decomposition, provide the following information:
--       o Name
--       o Abstract describing purpose of package.  For generic packages, this
--         shall include detailes of the capabilities provided by generic
--         parameters.
--       o Requirements trace
--       o Entities exported

--Local Entities
--  Describe the following entities which will be local to the TLCSC:
--    o Local data structures, encapsulated in the package body
--    o Files or data bases used by the TLCSC and not by any other TLCSC
--    o Data types defined local to the TLCSC and not used by any other
--      TLCSC
--    o Generic subprograms or packages defined local to the TLCSC and used
--      by entities exported by the TLCSC

--  Provide information describing the use of these local entities by other
--  entities within the TLCSC

--Additional "coding" information

--  o Security level -- None
--                      Confidential
--                      Secret

--  o Calling sequence

--  o History -- Prepared by
--               Baseline date

--  o Revision history -- Revised by
--                        Revision date
--                        Revision reason
--                        Brief description
```

(This page intentionally left blank.)

# SUPPLEMENTARY

# INFORMATION

DEPARTMENT OF THE AIR FORCE
WRIGHT LABORATORY (AFSC)
EGLIN AIR FORCE BASE, FLORIDA, 32542-5434

ERRATA
AD-B120253

REPLY TO ATTN OF: MNOI

13 Feb 92

SUBJECT: Removal of Distribution Statement and Export-Control Warning Notices

TO: Defense Technical Information Center
ATTN: DTIC/HAR (Mr William Bush)
Bldg 5, Cameron Station
Alexandria, VA 22304-6145

1. The following technical reports have been approved for public release by the local Public Affairs Office (copy attached).

| | Technical Report Number | AD Number |
|---|---|---|
| 1. | 88-18-Vol-4 | ADB 120 251 |
| 2. | 88-18-Vol-5 | ADB 120 252 |
| 3. | 88-18-Vol-6 | ADB 120 253 |
| 4. | 88-25-Vol-1 | ADB 120 309 |
| 5. | 88-25-Vol-2 | ADB 120 310 |
| 6. | 88-62-Vol-1 | ADB 129 568 |
| 7. | 88-62-Vol-2 | ADB 129 569 |
| 8. | 88-62-Vol-3 | ADB 129-570 |
| 9. | 85-93-Vol-1 | ADB 102-654 |
| 10. | 85-93-Vol-2 | ADB 102-655 |
| 11. | 85-93-Vol-3 | ADB 102-656 |
| 12. | 88-18-Vol-1 | ADB 120 248 |
| 13. | 88-18-Vol-2 | ADB 120 249 |
| 14. | 88-18-Vol-7 | ADB 120 254 |
| 15. | 88-18-Vol-8 | ADB 120 255 |
| 16. | 88-18-Vol-9 | ADB 120 256 |
| 17. | 88-18-Vol-10 | ADB 120 257 |
| 18. | 88-18-Vol-11 | ADB 120 258 |
| 19. | 88-18-Vol-12 | ADB 120 259 |

2. If you have any questions regarding this request call me at DSN 872-4620.

LYNN S. WARGO
Chief, Scientific and Technical
Information Branch

1 Atch
AFDTC/PA Ltr, dtd 30 Jan 92

ERRATA

DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AIR FORCE DEVELOPMENT TEST CENTER (AFSC)
EGLIN AIR FORCE BASE, FLORIDA 32542-5000

REPLY TO ATTN OF: PA (Jim Swinson, 882-3931) 30 January 1992

SUBJECT: Clearance for Public Release

TO: WL/MNA

The following technical reports have been reviewed and are approved for public release: AFATL-TR-88-18 (Volumes 1 & 2), AFATL-TR-88-18 (Volumes 4 thru 12), AFATL-TR-88-25 (Volumes 1 & 2), AFATL-TR-88-62 (Volumes 1 thru 3) and AFATL-TR-85-93 (Volumes 1 thru 3).

Virginia N. Pribyla

VIRGINIA N. PRIBYLA, Lt Col, USAF
Chief of Public Affairs

AFDTC/PA 92-039